intel

iRMX®
Extended I/O System Calls Reference Manual

Order Number: 462916-001

intel®

iRMX®
Extended I/O System Calls Reference Manual

Order Number: 462916-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are located directly after the reader reply card in the back of the manual.

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update or to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9 (a) (9).

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

Above	iLBX	iPSC	Plug-A-Bubble
BITBUS	i_m	iRMX	PROMPT
COMMputer	iMDDX	iSBC	Promware
CREDIT	iMMX	iSBX	QUEST
Data Pipeline	Insite	iSDM	QueX
Genius	int_el	iSSB	Ripplemode
ı	Intel376	iSXM	RMX/80
i	Intel386	Library Manager	RUPI
I^2ICE	int_elBOS	MCS	Seamless
ICE	Intelevision	Megachassis	SLD
iCEL	$int_eligent$ Identifier	MICROMAINFRAME	UPI
iCS	$int_eligent$ Programming	MULTIBUS	VLSiCEL
iDBP	Intellec	MULTICHANNEL	376
iDIS	Intellink	MULTIMODULE	386
	iOSP	OpenNET	386SX
	iPDS	ONCE	
	iPSB		

XENIX, MS-DOS, Multiplan, and Microsoft are trademarks of Microsoft Corporation. UNIX is a trademark of Bell Laboratories. Ethernet is a trademark of Xerox Corporation. Centronics is a trademark of Centronics Data Computer Corporation. Chassis Trak is a trademark of General Devices Company, Inc. VAX and VMS are trademarks of Digital Equipment Corporation. Smartmodem 1200 and Hayes are trademarks of Hayes Microcomputer Products, Inc. IBM, PC/XT, and PC/AT are registered trademarks of International Business Machines. Soft-Scope is a registered trademark of Concurrent Sciences.

REV.	REVISION HISTORY	DATE
-001	Original Issue.	02/89

PREFACE

INTRODUCTION

This manual documents the system calls of the Extended I/O System, one of the subsystems of the iRMX® I and iRMX II operating systems. The information provided in this manual is intended as a reference to the system calls and provides detailed descriptions of each call.

READER LEVEL

This manual is intended for programmers who are familiar with the concepts and terminology introduced in the *iRMX® I Nucleus User's Guide* or the *iRMX® II Nucleus User's Guide* and with the PL/M programming language.

CONVENTIONS

System call names appear as headings on the outside upper corner of each page. The first appearance of each system call name is printed in blue ink; subsequent appearances are in black.

Throughout this manual, system calls are shown using a generic shorthand (such as S$CREATE$FILE instead of RQSCREATE$FILE). This convention is used to allow easier alphabetic arrangement of the calls. The actual PL/M external-procedure names must be used in all calling sequences.

You can also invoke the system calls from assembly language, but you must obey the PL/M calling sequences when doing so. For more information on these calling sequences, refer to the *iRMX® I Programming Techniques Reference Manual* or the *iRMX® II Programming Techniques Reference Manual.*

CONTENTS

Chapter 1. iRMX® Extended I/O System Calls Page

Index

1 iRMX® EXTENDED I/O SYSTEM CALLS

1.1 INTRODUCTION

This manual describes the system calls provided by the iRMX® Extended I/O System. The manual contains:

- A brief explanation of condition codes.
- A system call dictionary listing the system calls by function.
- Complete descriptions of each system call.

> **iRMX I Note:** The information presented in this manual applies to both the iRMX I and iRMX II Operating Systems. However, a few of the system calls do not exist or operate differently in the iRMX I Operating System. These differences are described in note boxes such as this one.

Throughout this manual, PL/M data types, such as BYTE, WORD, and SELECTOR are used. In addition, the iRMX data type TOKEN is used. These words are always capitalized. If your compiler supports the SELECTOR data type, a TOKEN can be declared literally as SELECTOR. Because TOKEN is not a PL/M data type, you must declare it to be literally a SELECTOR every place you use it. Definitions of both PL/M and iRMX data types are given in the *iRMX® Extended I/O System User's Guide*. The word "token" in lowercase refers to a value that the operating system returns to a TOKEN (the data type) when it creates the object.

NOTE

The values NIL and SELECTOR$OF(NIL) are used throughout this manual. For the iRMX I Operating System, you may also use a value of zero in place of NIL and SELECTOR$OF(NIL). However, Intel recommends that you use NIL and SELECTOR$OF(NIL) in your iRMX I code to maintain upward compatibility with the iRMX II Operating System. For a description of the SELECTOR$OF and NIL built-in functions, refer to the PL/M-86 or PL/M-286 user's guides.

In each description of a system call, you will find a list of possible condition codes. This list is intended to help you debug your application system.

1.2 SYSTEM CALL DICTIONARY

The system call dictionary on the next few pages lists system calls by function rather than alphabetically.

The following abbreviations identify types of files for which a particular system call can be used:

PF means physical file
SF means stream file
NF means named file
ND means named directory

I/O JOBS		
Call	**Description**	**Page**
CREATEIOJOB	Creates an I/O job with a memory pool of up to 1M byte.	5
RQE$CREATE$IO$JOB	Creates an I/O job with a memory pool of up to 16M bytes. **This system call is not supported in the iRMX I Operating System.**	13
EXITIOJOB	Sends a message to a mailbox and deletes the calling task.	21
STARTIOJOB	Starts (makes ready) the initial task in an I/O job; the task was not started when the job was created.	35
LOGICAL NAMES		
HYBRID$DETACH$-DEVICE	Temporarily removes the correspondence between a logical name and a physical device established via LOGICAL$ATTACH$DEVICE.	28
LOGICAL$ATTACH$-DEVICE	Creates and catalogs a logical name for a device.	30
LOGICAL$DETACH$-DEVICE	Deletes a logical name created with LOGICAL$ATTACH$DEVICE.	33
S$CATALOG$-CONNECTION	Creates a logical name for a connection by cataloging the connection in the object directory of a specific job.	40

LOGICAL NAMES (continued)			
Call	**Description**		**Page**
SGETDIRECTORY$-ENTRY	Returns a directory entry name to the caller. **This system call is not supported in the iRMX I Operating System.**		74
SGETPATH$-COMPONENT	Returns the name of a named file as the file is known in its parent directory. **This system call is not supported in the iRMX I Operating System.**		85
S$LOOK$UP$-CONNECTION	Searches through an I/O job's object directories to find the connection associated with a logical name		87
S$UNCATALOG$-CONNECTION	Deletes a logical name from the object directory of a specific job.		142
CREATING FILES AND CONNECTIONS			
Call	**Description**	**Files**	**Page**
S$ATTACH$FILE	Creates a connection to an existing file.	All	36
S$CREATE$DI-RECTORY	Creates a new directory file.	ND	52
S$CREATE$FILE	Creates a new physical, stream, or named data file. It cannot create a named directory file.	PF,SF,NF	57
CHANGING ACCESS AND RENAMING			
S$CHANGE$ACCESS	Changes the access list for named file.	ND,NF	43
S$RENAME$FILE	Changes the path of a named file.	ND,NF	97
MANIPULATING DATA IN FILES			
S$CLOSE	Closes an open connection to a file.	All	50
S$OPEN	Opens a connection to a file.	All	89
S$READ$MOVE	Reads a number of bytes from a file to a buffer.	All	93

MANIPULATING DATA IN FILES (continued)			
Call	**Description**	**Files**	**Page**
S$SEEK	Moves the file pointer.	PF,ND,NF	102
S$TRUNCATE$FILE	Removes information from the end of a named data file.	NF	139
S$WRITE$MOVE	Writes a collection of bytes from a buffer to a file.	All	144
	DEVICES		
S$SPECIAL	Allows your task to perform functions pertaining to a specific device.	PF,SF	106
	OBTAINING STATUS		
GET$LOGICAL$-DEVICE$STATUS	Provides status information about logical devices.		23
SGETCON-NECTION$STATUS	Provides status information about file and device connections.	All	70
SGETFILE$STATUS	Allows a task to obtain information about a file.	All	76
	DELETING FILES AND CONNECTIONS		
S$DELETE$CON-NECTION	Deletes a file connection. It cannot delete a device connection.	All	63
S$DELETE$FILE	Deletes a stream, physical, or named file.	All	65
	USERS		
VERIFY$USER	Verifies a user's name and password.		148
GET$USER$IDS	Returns the user ID as defined in the User Definition File.		25

CREATEIOJOB creates an I/O job containing one task.

```
io$job = RQ$CREATE$IO$JOB(pool$min, pool$max, except$handler,
                          job$flags, task$priority, start$address,
                          data$seg, stack$ptr, stack$size,
                          task$flags, msg$mbox, except$ptr);
```

Input Parameters

pool$min

A WORD containing the minimum allowable size of the new job's pool, in 16-byte paragraphs. For example, a value of 35 indicates thirty-five 16-byte paragraphs. The Extended I/O System also uses this value as the initial size of the memory pool for the new job.

You must not assign pool$min a value less than 32. Furthermore, if the base of the stack$ptr parameter is equal to zero, you should ensure that pool$min is no less than 32 + (number of 16-byte paragraphs required to contain the stack). If you set pool$min to a value smaller than these minimums, the Extended I/O System will return an E$PARAM exceptional condition.

The purpose of the pool$min parameter in this system call is identical to the purpose of the pool$min parameter of the CREATE$JOB system call provided by the iRMX Nucleus. For additional information on the pool$min parameter, see the CREATE$JOB decription in the *iRMX® I Nucleus System Calls Reference Manual* or the *iRMX® II Nucleus System Calls Reference Manual*. For general information regarding memory pools, refer to the *iRMX® I Nucleus User's Guide* or the *iRMX® II Nucleus User's Guide*.

pool$max

A WORD containing the maximum allowable size of the new job's pool, in 16-byte paragraphs. For example, a value of 40 indicates forty 16-byte paragraphs.

You must set pool$max to a value no less than pool$min, or the Extended I/O System will return an E$PARAM exceptional condition.

The purpose of the pool$max parameter in this system call is identical to the purpose of the pool$max parameter of the CREATE$JOB system call provided by the iRMX Nucleus. For additional information on the pool$max parameter, see the CREATE$JOB decription in the *iRMX® I Nucleus System Calls Reference Manual* or the *iRMX® II Nucleus System Calls Reference Manual*. For general information regarding memory pools, refer to the *iRMX® I Nucleus User's Guide* or the *iRMX® II Nucleus User's Guide*.

except$handler — A POINTER to a structure of the following form:

```
DECLARE handler          STRUCTURE(
            exception$handler$offset    WORD,
            exception$handler$base      SELECTOR,
            exception$mode              BYTE);
```

The Extended I/O System expects you to designate an exception handler to be used as the new job's default exception handler. If you wish to designate the system default exception handler, you can do so by setting exception$handler$base to SELECTOR$OF(NIL). If you set the base to any other value, then the Extended I/O System assumes that the first two words of this structure point to the first instruction of your exception handler.

Set the exception$mode to tell the Extended I/O System when to pass control to the new task's exception handler. Encode the mode as follows:

Value	When Control Passes To Exception Handler
0	Control never passes to handler
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

For more information regarding exception handlers and exception modes, refer to the *iRMX® I Nucleus User's Guide* or the *iRMX® II Nucleus User's Guide*.

job$flags

A WORD that tells the Nucleus whether to check the validity of objects used as parameters in system calls. If bit 1 (where bit 0 is the low-order bit) is zero, the Nucleus will validate objects.

All bits other than bit 1 must be set to zero. This parameter serves precisely the same purpose as the job$flags parameter of the CREATE$JOB system call provided by the Nucleus. For additional information on the job$flags parameter, see the CREATE$JOB decription in the *iRMX® I Nucleus System Calls Reference Manual* or the *iRMX® II Nucleus System Calls Reference Manual.*

task$priority

A BYTE which establishes the priority of the initial task in the new job.

- If equal to zero, specifies that the new job's initial task is to have a priority equal to the the maximum priority of the initial job of the Extended I/O System. For more information about the initial job of the Extended I/O System, refer to the configuration reference manual included with the operating system.
- If not equal to zero, contains the priority of the initial task of the new job. If this priority is higher than (numerically less than) the maximum priority of the initial job of the Extended I/O System, an E$PARAM error occurs.

start$address

A POINTER to the first instruction of the code segment for the new job's initial task. This code segment can be, but is not required to be, an iRMX segment.

data$seg

A SELECTOR which,

- if SELECTOR$OF(NIL), indicates one of two things. Either the new job's initial task uses no data segment, or it creates one for itself. Tasks can create their own data segments only under special circumstances. To find out more about these circumstances, refer to the Extended I/O System parameters section of the configuration reference manual included with the operating system.
- if not SELECTOR$OF(NIL), contains the base address of the data segment of the new job's initial task. This data segment can be, but is not required to be, an iRMX segment.

stack$ptr

A POINTER which,

- if the stack pointer is NIL, specifies that the Nucleus should allocate a stack for the new job's initial task. The length of the allocated stack is determined by the stack$size parameter of this system call. Be aware that this stack is not an iRMX segment.
- if the stack pointer is not equal to NIL, points to the base of the stack for the new job's initial task. Because the Nucleus does not allocate this stack, you must allocate it during the configuration process, or your application code must allocate it while the system is running.

stack$size

A WORD containing the size, in bytes, of the stack for the new job's initial task. If you specify less than 200, the Extended I/O System will increase the size to 200. For information regarding the amount of stack to allocate, refer to the discussion of stack sizes in the *iRMX® I Programming Techniques Reference Manual* or the *iRMX® II Programming Techniques Reference Manual.*

If you are allocating the stack during configuration, or if the application code is allocating the stack while the system is running, the value of this parameter will be the precise amount of stack that the system can use. However, if the Nucleus is allocating the stack for you, it might allocate as many as 15 additional bytes in order to make the stack occupy whole 16-byte paragraphs.

task$flags

A WORD in which all bits except the two low-order bits must be set to zero. The upper 14 bits are reserved for Intel's use.

Bit Zero: Use the low-order bit (bit 0) to tell the operating system whether the new job's initial task uses floating-point instructions. A value of 1 indicates the presence of floating-point instructions, while a zero indicates the absence of floating-point instructions.

Bit One: Bit 1 indicates whether the initial task in the job should run immediately, or whether it should wait until a STARTIOJOB system call is issued to start it. Set bit 1 to zero if the task is to be made ready to run; set bit 1 to one if the task is to wait until the STARTIOJOB call is issued.

msg$mbox

A TOKEN for a mailbox. When a task exits (by invoking EXITIOJOB), the Extended I/O System sends a message to this mailbox. If you desire no such message, assign msg$mbox a value of SELECTOR$OF(NIL).

The format of the message is as follows:

```
DECLARE message STRUCTURE(
            termination$code    WORD,
            user$fault$code     WORD,
            job$token           TOKEN,
            return$data$len     BYTE,
            return$data(*)      BYTE);
```

where:

termination$code — A WORD that indicates why an I/O job terminated, as follows:

CODE	MEANING
0	Some task within the job--the terminating task--invoked the EXITIOJOB system call, and indicated with this code that no problem caused the termination. The job has not yet been deleted, and some of its tasks might still be ready.
1	The job was deleted because some task invoked the DELETE$JOB system call.
any other code	Some task within the job invoked the EXITIOJOB system call and indicated that the job was terminated because some problem occurred. The job has not yet been deleted and some of its tasks might still be ready.

user$fault$code — A WORD that contains an encoded reason for termination of the new job. Whenever the termination$code has a value other than 0 or 1, this parameter contains an error code that the terminating task specified when invoking the EXITIOJOB system call. The precise meaning of this code is provided by the terminating task, not by the operating system.

job$token — A TOKEN for the job that was terminated.

return$data$len — A BYTE that specifies the length (in bytes) of the return$data parameter described below. The maximum length is 89 (decimal) bytes.

return$data	A sequence of BYTES that contain data specified by the terminating task when it invoked the EXITIOJOB system call.

Output Parameters

io$job	A TOKEN that represents the newly created job. The operating system returns a valid token only if the Extended I/O System returns an E$OK condition code.
except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

This system call creates a job whose tasks can invoke the system calls provided by the Extended I/O System. Such jobs are called I/O jobs, and they differ from other jobs in these ways:

- Job parameter defaults: Many of the parameters required by the Nucleus's CREATE$JOB system call are not required by the CREATEIOJOB system call. These parameters include

  directory$size
  param$object
  max$objects
  max$tasks
  max$priority

  The Extended I/O System allows you to specify values for some of these parameters during the system configuration process. The precise instructions for defining these values are provided in the configuration reference manual included with the operating system.

- Default job attributes: The CREATEIOJOB system call provides default values for the following I/O job attributes:

  global job
  default user
  default prefix

  The values for these attributes are passed from parent job to child job. For instance, if Job A uses the CREATEIOJOB system call to spawn Job B, then the Extended I/O System copies the values of the Job A attributes into the Job B attributes. Be aware that if you change the Job A attributes after Job B has been created, the changed values are not copied into Job B.

You can set the values for these attributes for the "first parent" job during the process of configuring your system.

- Notification of job termination: The CREATEIOJOB system call provides a mechanism for notifying the parent job of the termination of the I/O job. The Extended I/O System implements this mechanism by sending a termination message to a mailbox of your choice whenever a task in the I/O job terminates (calls EXITIOJOB). You specify the mailbox by using the msg$mbox parameter of this system call.

The CREATEIOJOB system call can be called only from another I/O job. You can set up one or more initial I/O jobs while configuring the operating system. For more information about configuration, refer to the configuration reference manual included with the operating system.

Do not delete a task in an I/O job if the task is using a connection (that is, if the connection has not been deleted). If you do so, the connection will not be available to any other task.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
E$EXIST	0006H	At least one of the following is true: • The token cataloged under the name RQGLOBAL (the global job) is not a token for an existing object. (See the *iRMX® Basic I/O System User's Guide* for information on the global object directory.) • The value assigned to the msg$mbox parameter is not a token for an existing mailbox. • The user TOKEN is not valid.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
E$NOUSER	8021H	The calling task's job does not have a default user, or the object cataloged under the logical name R?IOUSER is not a user object. (See the *iRMX® Basic I/O System User's Guide* for information on R?IOUSER.)

E$PARAM	8004H	At least one of the following is true: • The value assigned to the pool$min parameter is less than 32 decimal, or it is greater than the value assigned to the pool$max parameter. • The value assigned to task$priority is not zero and is greater than (numerically less than) the maximum priority of the calling I/O job. • The value assigned to the exception$mode parameter is outside the range 0-3, inclusive.
EIOJOB	0047H	The calling task's job is not an I/O job.

E$CREATE$IO$JOB creates an I/O job containing one task with a maximum of 16M bytes of memory pool.

> **iRMX I Note:** The RQE$CREATE$IO$JOB system call is not supported in the iRMX I Operating System.

```
io$job = RQE$CREATE$IO$JOB(pool$min, pool$max, except$handler,
                           job$flags, task$priority, start$address,
                           data$seg, stack$ptr, stack$size,
                           task$flags, msg$mbox, except$ptr);
```

Input Parameters

pool$min

A DWORD containing the minimum allowable size of the new job's pool, in 16-byte paragraphs. For example, a value of 35 indicates thirty-five 16-byte paragraphs. The Extended I/O System also uses this value as the initial size of the memory pool for the new job. The memory initially allocated is always contiguous. If additional memory is requested, it is not necessarily contiguous.

You must not assign pool$min a value less than 32. Furthermore, if the base of the stack$ptr parameter is equal to zero, you should ensure that pool$min is no less than 32 + (number of 16-byte paragraphs required to contain the stack). If you set pool$min to a value smaller than these minimums, the Extended I/O System will return an E$PARAM exceptional condition.

The purpose of the pool$min parameter in this system call is identical to the purpose of the pool$min parameter of the RQE$CREATE$JOB system call provided by the iRMX II Nucleus. For additional information on the pool$min parameter, see the RQE$CREATE$JOB description in the *iRMX® II Nucleus System Calls Reference Manual.* For general information regarding memory pools, refer to the *iRMX® II Nucleus User's Guide.*

pool$max

A DWORD containing the maximum allowable size of the new job's pool, in 16-byte paragraphs. For example, a value of 40 indicates forty 16-byte paragraphs.

You must set pool$max to a value no less than pool$min, or the Extended I/O System will return an E$PARAM exceptional condition.

The purpose of the pool$max parameter in this system call is identical to the purpose of the pool$max parameter of the RQE$CREATE$JOB system call provided by the iRMX II Nucleus. For additional information on the pool$max parameter, see the RQE$CREATE$JOB description in the *iRMX® II Nucleus System Calls Reference Manual.* For general information regarding memory pools, refer to the *iRMX® II Nucleus User's Guide.*

except$handler

A POINTER to a structure of the following form:

```
DECLARE handler     STRUCTURE(
        exception$handler$offset    WORD,
        exception$handler$base      SELECTOR,
        exception$mode              BYTE)
```

The Extended I/O System expects you to designate an exception handler to be used as the new job's default exception handler. If you wish to designate the system default exception handler, you can do so by setting exception$handler$base to SELECTOR$OF(NIL). If you set the base to any other value, then the Extended I/O System assumes that the first two words of this structure point to the first instruction of your exception handler.

Set the exception$mode to tell the Extended I/O System when to pass control to the new task's exception handler. Encode the mode as follows:

Value	When Control Passes To Exception Handler
0	Control never passes to handler
1	On programmer errors only
2	On environmental conditions only
3	On all exceptional conditions

For more information regarding exception handlers and exception modes, refer to the *iRMX® II Nucleus User's Guide.*

job$flags

A WORD that tells the Nucleus whether to check the validity of objects used as parameters in system calls. If bit 1 (where bit 0 is the low-order bit) is zero, the Nucleus will validate objects.

All bits other than bit 1 must be set to zero. This parameter serves precisely the same purpose as the job$flags parameter of the CREATE$JOB system call provided by the Nucleus. For additional information on the job$flags parameter, see the CREATE$JOB decription in the *iRMX® II Nucleus System Calls Reference Manual.*

task$priority

A BYTE which establishes the priority of the initial task in the new job.

- If equal to zero, specifies that the new job's initial task is to have a priority equal to the the maximum priority of the initial job of the Extended I/O System. For more information about the initial job of the Extended I/O System, refer to the configuration reference manual included with the operating system.
- If not equal to zero, contains the priority of the initial task of the new job. If this priority is higher than (numerically less than) the maximum priority of the initial job of the Extended I/O System, an E$PARAM error occurs.

start$address

A POINTER to the first instruction of the code segment for the new job's initial task. This code segment can be, but is not required to be, an iRMX segment.

data$seg

A SELECTOR which,

- if SELECTOR$OF(NIL), indicates one of two things. Either the new job's initial task uses no data segment, or it creates one for itself. Tasks can create their own data segments only under special circumstances. To find out more about these circumstances, refer to the Extended I/O System parameters section of the configuration reference manual included with the operating system.
- if not SELECTOR$OF(NIL), contains the base address of the data segment of the new job's initial task. This data segment can be, but is not required to be, an iRMX segment.

stack$ptr

A POINTER which,

- if the stack pointer is NIL, specifies that the Nucleus should allocate a stack for the new job's initial task. The length of the allocated stack is determined by the stack$size parameter of this system call. Be aware that this stack is not an iRMX segment.
- if the stack pointer is not equal to NIL, points to the base of the stack for the new job's initial task. Because the Nucleus does not allocate this stack, you must allocate it during the configuration process, or your application code must allocate it while the system is running.

stack$size

A WORD containing the size, in bytes, of the stack for the new job's initial task. If you specify less than 200, the Extended I/O System will increase the size to 200. For information regarding the amount of stack to allocate, refer to the chapter of the *iRMX® II Programming Techniques* manual that discusses stack sizes.

If you are allocating the stack during configuration, or if the application code is allocating the stack while the system is running, the value of this parameter will be the precise amount of stack that the system can use. However, if the Nucleus is allocating the stack for you, it might allocate as many as 15 additional bytes in order to make the stack occupy whole 16-byte paragraphs.

task$flags

A WORD in which all bits except the two low-order bits are set to zero.

Bit Zero: Use the low-order bit (bit 0) to tell the operating system whether the new job's initial task uses floating-point instructions. A value of 1 indicates the presence of floating-point instructions, while a zero indicates the absence of floating-point instructions.

Bit One: Bit 1 indicates whether the initial task in the job should run immediately, or whether it should wait until a STARTIOJOB system call is issued to start it. Set bit 1 to zero if the task is to be made ready to run; set bit 1 to one if the task is to wait until the STARTIOJOB call is issued.

msg$mbox

A TOKEN for a mailbox. When a task exits (by invoking EXITIOJOB), the Extended I/O System sends a message to this mailbox. If you desire no such message, assign msg$mbox a value of zero.

The format of the message is as follows:

```
DECLARE message     STRUCTURE(
                  termination$code     WORD,
                  user$fault$code      WORD,
                  job$token            TOKEN,
                  return$data$len      BYTE,
                  return$data(*)       BYTE);
```

where:

termination$code — A WORD that indicates why an I/O job terminated, as follows:

CODE	MEANING
0	Some task within the job--the terminating task--invoked the EXITIOJOB system call, and indicated with this code that no problem caused the termination. The job has not yet been deleted, and some of its tasks might still be ready.
1	The job was deleted because some task invoked the DELETE$JOB system call.
any other code	Some task within the job invoked the EXITIOJOB system call and indicated that the job was terminated because some problem occurred. The job has not yet been deleted and some of its tasks might still be ready.

user$fault$code — A WORD that contains an encoded reason for termination of the new job. Whenever the termination$code has a value other than 0 or 1, this parameter contains an error code that the terminating task specified when invoking the EXITIOJOB system call. The precise meaning of this code is provided by the terminating task, not by the operating system.

job$token — A TOKEN for the job that was terminated.

return$data$len — A BYTE that specifies the length (in bytes) of the return$data parameter described below. The maximum length is 89 (decimal) bytes.

return$data	A sequence of BYTES that contain data specified by the terminating task when it invoked the EXITIOJOB system call.

Output Parameters

io$job	The TOKEN that represents the newly created job. The operating system returns a valid token only if the Extended I/O System returns an E$OK condition code.
except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

This system call creates a job whose tasks can invoke the system calls provided by the Extended I/O System. Such jobs are called I/O jobs, and they differ from other jobs in these ways:

- Job parameter defaults: Many of the parameters required by the Nucleus's CREATE$JOB system call are not required by the E$CREATE$IO$JOB system call. These parameters include

  directory$size
  param$object
  max$objects
  max$tasks
  max$priority

  The Extended I/O System allows you to specify values for some of these parameters during the system configuration process. The precise instructions for defining these values are provided in the configuration reference manual included with the operating system.

- Default job attributes: The E$CREATE$IO$JOB system call provides default values for the following I/O job attributes:

  global job
  default user
  default prefix

  The values for these attributes are passed from parent job to child job. For instance, if Job A uses the E$CREATE$IO$JOB system call to spawn Job B, then the Extended I/O System copies the values of the Job A attributes into the Job B attributes. Be aware that if you change the Job A attributes after Job B has been created, the changed values are not copied into Job B.

  You can set the values for these attributes for the "first parent" job during the process of configuring your system.

- Notification of job termination: The E$CREATE$IO$JOB system call provides a mechanism for notifying the parent job of the termination of the I/O job. The Extended I/O System implements this mechanism by sending a termination message to a mailbox of your choice whenever a task in the I/O job terminates (calls EXITIOJOB). You specify the mailbox by using the msg$mbox parameter of this system call.

The E$CREATE$IO$JOB system call can be called only from another I/O job. You can set up one or more initial I/O jobs while configuring the operating system. For more information about configuration, refer to the configuration reference manual included with the operating system.

Do not delete a task in an I/O job if the task is using a connection (that is, if the connection has not been deleted). If you do so, the connection will not be available to any other task.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
E$EXIST	0006H	At least one of the following is true: • The token cataloged under the name RQGLOBAL (the global job) is not a token for an existing object. (See the *iRMX® Basic I/O System User's Guide* for information on the global object directory.) • The value assigned to the msg$mbox parameter is not a token for an existing mailbox.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
E$NOUSER	8021H	The calling task's job does not have a default user, or the object cataloged under the logical name R?IOUSER is not a user object. (See the *iRMX® Basic I/O System User's Guide* for information on R?IOUSER.)

E$PARAM	8004H	At least one of the following is true: • The value assigned to the pool$min parameter is less than 32 decimal, or it is greater than the value assigned to the pool$max parameter. • The value assigned to task$priority is not zero and is greater than (numerically less than) the maximum priority of the calling I/O job. • The value assigned to the exception$mode parameter is outside the range 0-3, inclusive.
EIOJOB	0047H	The calling task's job is not an I/O job.

EXITIOJOB sends a message to a previously designated mailbox and deletes the calling task.

```
CALL RQ$EXIT$IO$JOB(user$fault$code, return$data$ptr, except$ptr);
```

Input Parameters

user$fault$code	A WORD containing the encoded reason for terminating the job. If you terminate the job under normal circumstances, you should enter a value of zero. If you terminate the job because of a problem, you should enter an error code that identifies the problem. The Extended I/O System sends a structure containing the value you enter to the mailbox specified in the CREATEIOJOB system call.
return$data$ptr	A POINTER to a buffer containing a STRING containing data (provided by the calling task) to be returned to the message mailbox specified in the CREATEIOJOB system call. If you enter NIL, no data is returned. If the string is longer than 89 (decimal) bytes, only the first 89 bytes are returned.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

The EXITIOJOB system call complements the CREATEIOJOB system call. Using the EXITIOJOB system call, a task can delete itself and have the Extended I/O System notify the parent job of the deletion.

When a task in an I/O job (a job created by the CREATEIOJOB system call) invokes the EXITIOJOB system call, two things happen:

- The Extended I/O System deletes the task (but not the job containing the task) that invoked the EXITIOJOB system call.
- The Extended I/O System sends a termination message to the mailbox specified in the CREATEIOJOB system call.

Special Circumstances

Your application code can use this system call to bring about an orderly deletion of an I/O job. To do this, have a task within the I/O job invoke this system call. Then have a task in the parent job receive the message and delete the I/O job. Under certain circumstances, this system call does not delete the calling task or does not send a termination message.

Calling Task Not Deleted: Although the EXITIOJOB system call generally deletes the calling task, this deletion does not occur in the following circumstances:

- If the DELETE$TASK system call (which the Extended I/O System calls) returns an exception code to the Extended I/O System.
- If the calling task is an interrupt task.

In both cases, the Extended I/O System returns control to the calling task and issues an exceptional condition code to indicate the nature of the problem. Under any other circumstance, the Extended I/O System deletes the calling task.

Termination Message Not Sent: Even if it fails to delete the task, the Extended I/O System sends the termination message if one has been requested, except for the following circumstances:

- If the msg$mbox parameter of the CREATEIOJOB was set to SELECTOR$OF(NIL).
- If the mailbox specified in the msg$mbox parameter of the CREATEIOJOB system call no longer exists.

Condition Codes

E$CONTEXT	0005H	The task invoking the EXITIOJOB system call is an interrupt task and cannot be deleted.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.

The GET$LOGICAL$DEVICE$STATUS system call provides status information about a logical device.

```
CALL RQ$GET$LOGICAL$DEVICE$STATUS(log$name$ptr, dev$info$ptr,
                                  except$ptr);
```

Input Parameter

log$name$ptr — A POINTER to a STRING containing the logical name under which the logical device object is cataloged in the root object directory.

Output Parameters

dev$info$ptr — A POINTER to a structure in which the Extended I/O System returns the status information. You can allocate memory for this structure by requesting an iRMX segment or by reserving the memory in your code. The structure must have the following form:

```
DECLARE dev$info    STRUCTURE(
                        device$name(15)     BYTE,
                        file$driver         BYTE,
                        num$conns           WORD,
                        owner$id            WORD);
```

where:

device$name — A STRING containing the physical name associated with the device. This is the name established during Basic I/O System configuration.

file$driver — The file driver associated with the device. Possible values include

Value	File Driver
1	physical
2	stream
4	named
5	remote

num$conns — The current number of connections to the device.

	owner$id	The owner ID for this device. This ID is the first ID listed in the default user object of the attaching task's job.
except$ptr	A POINTER to a WORD in which the Extended I/O System returns the condition code.	

Description

The GET$LOGICAL$DEVICE$STATUS system call allows a task to obtain status information about logical names that represent devices. The Extended I/O System does not check access before returning status information.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$EXIST	0006H	The device connection corresponding to the logical name is being deleted.
E$LIMIT	0004H	Either the user object or the calling task's job is already involved in 255 (decimal) I/O operations.
ELOGNAME$-NEXIST	0045H	The logical name was not found in the root object directory.
ELOGNAME$-SYNTAX	0040H	The syntax of the specified logical name is incorrect because at least one of the following conditions is true: • The name was missing matching colons (:). • The STRING pointed to by the log$name$ptr parameter is of zero length or has a length greater than 12 (not including colons (:)). • The logical name contains invalid characters.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTDEVICE	8041H	The specified logical name does not represent a valid device connection.

The GET$USER$IDS system call returns the user ID(s) associated with a USER defined in the User Definition File (UDF).

```
CALL RQ$GET$USER$IDS(name$ptr, ids$ptr, except$ptr);
```

Input Parameter

name$ptr — A POINTER to a STRING containing the user name. (Only the first eight characters are significant.)

Output Parameters

ids$ptr — A POINTER to a structure where the ID(s) associated with the user name will be placed. The structure has the following form:

```
DECLARE ids     STRUCTURE (
                    length      WORD,
                    count       WORD,
                    id(*)       WORD);
```

where:

length — Should be set by the caller to the maximum number of ID(s) desired.

count — Will contain the number of valid IDs in the ID array after GET$USER$IDS has returned to the caller. This value will never be greater than the ids.length. The user does not need to initialize this value.

id — Is an array of IDs obtained from the UDF. The length of this array is contained in ids.count. The user does not need to initialize this array.

except$ptr — A POINTER to a WORD where the Extended I/O System returns a condition code.

Description

This system call returns the user ID(s) associated with a user name defined in the User Definition File (UDF). It searches the file :CONFIG:UDF for the user name pointed to by the name$ptr parameter and if found, returns that user's ID(s). For details, refer to the I/O Users screen description in the configuration reference manual included with the operating system.

Condition Codes

E$OK	0000H	No exceptional conditions.
EBADCALL	8005H	A task wrote over the interface library or over the EIOS job.
E$CONTEXT	0005H	The calling job is not an I/O job.
EDEVDETACHING	0039H	An I/O operation could not be performed on the device (:SD:) because it was being detached.
E$DEVFD	0022H	The device (:SD:) cannot be used with the file driver as specified in the preceding logical attach operation.
EUDFFORMAT	0048H	The UDF is not in the correct format.
E$FACCESS	0026H	The user does not have the proper access rights for the requested operation.
E$FLUSHING	002CH	The device (:SD:) is being detached.
E$FNEXIST	0021H	At least one of the following is true: • The file or a file in its path does not exist. • The specified physical device was not found.
E$FTYPE	0027H	A path component is not a directory file.
E$ILLVOL	002DH	The file driver given in the volume label conflicts with the file driver specified in the preceding logical attach operation.
E$INVALID$FNODE	003DH	The fnode associated with a file is either marked not allocated, or the fnode number is out of range.
EIOHARD	0052H	A hard error occurred; the BIOS cannot retry the request.
EIOMEM	0042H	The BIOS job did not have enough memory to perform the requested function.
EIOOPRINT	0053H	The device is off-line; operator intervention is required.

EIOSOFT	0051H	A soft error occurred and the BIOS has retried the operation and has failed; a retry is not possible.
EIOUNCLASS	0050H	An unclassified I/O error occurred.
EIOWR$PROT	0054H	The volume specified in this call is write protected.
E$LIMIT	0004H	The root job object directory is full.
ELOGNAME$-NEXIST	0045H	The logical name was not found in the caller's object directory, the global job object directory, or the root job object directory.
E$MEDIA	0044H	The device associated with the system call is off-line.
E$NAME$NEXIST	0049H	The name specified in this call is not defined.
E$NOPREFIX	8022H	The caller's job does not have a default prefix, or it is invalid.
E$NOUSER	8021H	The caller's job does not have a default user or it is invalid.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
E$PARAM	8004H	At least one of the following is true: • The name$ptr parameter is equal to NIL. • The length field of the ids structure is equal to zero. • The name contains invalid characters.
E$SHARE	0028H	The file is not sharable with the requested access.

The HYBRID$DETACH$DEVICE system call removes the correspondence between a logical name and a physical device without removing the logical name from the root object directory.

```
CALL RQ$HYBRID$DETACH$DEVICE(log$name$ptr, except$ptr);
```

Input Parameter

log$name$ptr — A POINTER to a STRING containing the logical name under which the logical device object is cataloged in the root object directory.

Output Parameter

except$ptr — A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

HYBRID$DETACH$DEVICE removes the linkage between a logical name and a physical device without removing the logical name from the root object directory. When a task calls HYBRID$DETACH$DEVICE, the Extended I/O System detaches the device by issuing the Basic I/O System A$PHYSICAL$DETACH$DEVICE call. In so doing, the Extended I/O System specifies the hard detach option which deletes all connections to files on the device.

A device detached using HYBRID$DETACH$DEVICE can be reattached in one of two ways:

- A task can issue the Basic I/O System A$PHYSICAL$ATTACH$DEVICE system call.
- A task can use the device's logical name (which is still cataloged in the root object directory) as the prefix portion of a path name when issuing an Extended I/O System call. In this case, the Extended I/O System physically attaches the device using the parameters originally specified when the logical name was established (via LOGICAL$ATTACH$DEVICE).

A task cannot use LOGICAL$ATTACH$DEVICE to reattach a device that HYBRID$DETACH$DEVICE detached. Before reattaching a device with LOGICAL$ATTACH$DEVICE, a task must first issue LOGICAL$DETACH$DEVICE.

The HYBRID$DETACH$DEVICE system call is particularly useful for tasks that must temporarily detach a device and attach it in a different way (for example, attaching a disk as a physical device when formatting a volume). These tasks can call HYBRID$DETACH$DEVICE to detach the device, attach the device using A$PHYSICAL$ATTACH$DEVICE, perform the special processing on the device, and detach the device using A$PHYSICAL$DETACH$DEVICE. Later, when a task includes the device's logical name in an Extended I/O System call, the Extended I/O System automatically reattaches the device in the previous manner.

The HYBRID$DETACH$DEVICE system call can be issued as follows:

- By the task ("attaching task") that created the logical name by issuing LOGICAL$ATTACH$DEVICE, or by some other task in the same job as the attaching task.
- By any task in a job whose default user object contains the file's owner ID in its ID list.
- By the System Manager.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$EXIST	0006H	The device connection corresponding to the logical name is being deleted.
E$LIMIT	0004H	Either the user object or the calling task's job is already involved in 255 (decimal) I/O operations.
ELOGNAME$-NEXIST	0045H	The logical name was not found in the root object directory.
ELOGNAME$-SYNTAX	0040H	The syntax of the specified logical name is incorrect because at least one of the following conditions is true: • The STRING pointed to by the log$name$ptr parameter is of zero length, has a length greater than 12 not including colons (:), or is missing matching colons. • The logical name contains invalid characters.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTDEVICE	8041H	The specified logical name does not represent a valid device connection.
ENOTOWNER	0046H	The user (specified by the default user object) is not the user that attached the device.

The LOGICAL$ATTACH$DEVICE system call assigns a logical name to a physical device.

CAUTION

Any task that uses this system call loses its device independence. To maintain as much device independence as possible in your application, a few selected tasks should perform all attaching and detaching of devices.

```
CALL RQ$LOGICAL$ATTACH$DEVICE(log$name$ptr, dev$name, file$driver,
                              except$ptr);
```

Input Parameters

log$name$ptr — A POINTER to a STRING (of 1 to 12 characters) containing the logical name to be assigned to a device. The name can be delimited with colons (:). The operating system removes the colons so that a logical name with colons is the same as one without (e.g., :F0: is effectively the same as F0), and colons do not count in the length of the name. If you intend to use this logical name as part of a path name in other system calls, enclose it in colons.

dev$name — A POINTER to a STRING containing the name of the device to which the logical name is assigned. This device name is the name of a Device-Unit Information Block (DUIB) specified during Basic I/O System configuration.

file$driver — A BYTE specifying which Basic I/O System file driver to use with the device. Possible values are as follows:

Value	File Driver
1	physical
2	stream
4	named
5	remote

Output Parameter

except$ptr — A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

LOGICAL$ATTACH$DEVICE assigns a logical name to a physical device. This system call creates a Logical Device Object that corresponds to a physical device. This Logical Device Object is cataloged in the root object directory under the logical name pointed to by log$name$ptr. The Logical Device Object must be cataloged before the Extended I/O System can make connections to files on the device.

The first Extended I/O System call that uses the logical name as a prefix in a path name causes the physical device to be attached. (The Extended I/O System uses the Basic I/O System call A$PHYSICAL$ATTACH$DEVICE.) The logical name can be used as a prefix in other system calls and can be deleted by LOGICAL$DETACH$DEVICE.

If the first attempt to attach the device fails, the Extended I/O System will try again if you select the retry feature during configuration. You select the retry feature by specifing a non-zero value for the "(RPA) Retries on Physical Attachdevice" parameter on the EIOS screen of the Interactive Configuration Utility. The Extended I/O System will continue trying to attach the device until the device is attached successfully or the Extended I/O System has retried the number of times specified in the RPA parameter, whichever comes first.

> **iRMX I Note:** The iRMX I Interactive Configuration Utility does not support the "(RPA) Retries on Physical Attachdevice" parameter.

Because of the nature of LOGICAL$ATTACH$DEVICE, some exception codes that result because of errors in this system call are not returned until the Extended I/O System tries to attach the device with A$PHYSICAL$ATTACH$DEVICE.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONTEXT	0005H	The root object directory already contains an entry with the name pointed to by the log$name$ptr parameter.
E$LIMIT	0004H	At least one of the following is true: • The calling task's job object directory is full. • The root object directory is full. • The calling task's job is not an I/O job.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete this call.

ELOGNAME$-SYNTAX	0040H	The specified logical name is syntactically incorrect because at least one of the following conditions is true: • The STRING pointed to by the log$name$ptr parameter is of zero length or has a length of greater than 12 (including the colons). • The logical name contains invalid characters.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.

The LOGICAL$DETACH$DEVICE system call removes the correspondence between a logical name and a physical device, and removes the logical name from the root object directory.

```
CALL RQ$LOGICAL$DETACH$DEVICE(log$name$ptr, except$ptr);
```

Input Parameter

log$name$ptr	A POINTER to a STRING containing the logical name under which the logical device object is catalogued in the root object directory.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

LOGICAL$DETACH$DEVICE severs an association created by a call to LOGICAL$ATTACH$DEVICE and deletes the corresponding entry in the root object directory. After LOGICAL$DETACH$DEVICE is issued, users cannot create new connections using the logical name as a prefix. When the last file connection on the physical device is severed, the Extended I/O System detaches the device (issues the Basic I/O System call A$PHYSICAL$DETACH$DEVICE).

The LOGICAL$DETACH$DEVICE system call can be issued as follows:

- By the task ("attaching task") that created the logical name by issuing LOGICAL$ATTACH$DEVICE, or by some other task in the same job as the attaching task.
- By another job having the same owner ID in its default user object.
- By the System Manager.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$EXIST	0006H	The device connection corresponding to this logical name is being deleted.

E$LIMIT	0004H	One of the following is true: • The job has reached the object limit of the calling task's object directory. • Either the user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job.
ELOGNAME$-NEXIST	0045H	The logical name was not found in the root object directory.
ELOGNAME$-SYNTAX	0040H	The syntax of the specified logical name is incorrect because at least one of the following conditions is true: • The STRING pointed to by the log$name$ptr parameter is of zero length or has a length greater than 12 (not including colons (:)). • The logical name contains invalid characters.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTDEVICE	8041H	The specified logical name does not represent a valid device connection.
ENOTOWNER	0046H	The user (specified by the default user object) is not the user that attached the device.

STARTIOJOB starts the execution of a task in an I/O job. The task was not started when the I/O job was created.

```
CALL RQ$START$IO$JOB(io$job, except$ptr);
```

Input Parameter

io$job	TOKEN for the I/O job to be started. This is the TOKEN that was returned by the call to CREATEIOJOB.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

When you call CREATEIOJOB you can specify (with the task$flags parameter) that the task in the new job not start running until the STARTIOJOB call is issued. In this way you can initialize any items that need to be set before the initial task in the new job starts running. For example, you can create a job, catalog a logical name in the new job's object directory, and then issue STARTIOJOB.

Condition Codes

E$OK	0000H	No exceptional conditions.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
E$TIME	0001H	The job cannot be started yet, probably because the operating system has not finished processing the CREATEIOJOB call that created this job.

S$ATTACH$FILE

The S$ATTACH$FILE system call creates a connection to an existing file.

```
connection = RQ$S$ATTACH$FILE(path$ptr, except$ptr);
```

Input Parameter

path$ptr	A POINTER to a STRING containing the path name of the file to be attached.

Output Parameters

connection	The TOKEN that represents the new connection to the file.
except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

This system call allows a task to obtain a connection to any named, physical, or stream file.

The Extended I/O System allows any task to attach any file. However, if the file being attached is a named file, the Extended I/O System computes access rights for the connection. These access rights are based on the file's access list and the user IDs in the default user object of the calling task's job. (Refer to the iRMX Operating System user guides for more information.) If the file's access list allows no access to the users listed in the default user object, the call creates the connection, but it allows no access.

Special Considerations for iRMX®-NET

Unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$ALREADY$-ATTACHED	0038H	The Extended I/O System cannot attach the device containing the file because the Basic I/O system has already attached the device.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
EDEVDETACHING	0039H	The device containing the specified file is in the process of being detached.
E$DEVFD	0022H	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. In the process, it found that the device and the device driver specified in the logical attachment were incompatible.
E$EXIST	0006H	The device connection TOKEN is invalid.
E$FACCESS	0026H	The default user object is not allowed access to the file. See the Description section for more information.
E$FNEXIST	0021H	A file in the specified path, or the target file itself, does not exist or is marked for deletion.
E$FTYPE	0027H	The specified path is attempting to use a data file as a directory.
E$ILLVOL	002DH	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. During this process, it examined the volume label and found that the volume does not contain named files. This prevented the Extended I/O System from completing physical attachment because the named file driver was requested during logical attachment.
E$INVALID$FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; you should delete it.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOMEM	0042H	The BIOS job did not have enough memory to perform the requested function.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.

EIOSOFT	0051H	A soft I/O error occurred. The Basic I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
E$LIMIT	0004H	At least one of the following is true: • The calling task reached the object limit. • The user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*.
ELOGNAME$-NEXIST	0045H	The specified path contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • The specified path contains a logical name that is either longer than 12 characters (including colons), has no characters, or contains invalid characters.
E$MEDIA	0044H	The device containing the specified file is not on-line.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E$NAME$NEXIST	0049H	The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF).
E$NOPREFIX	8022H	You did not specify an explicit prefix (logical name), and the default prefix for the calling task's job is either undefined, or it is not a valid device connection or file connection.

ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTLOG$NAME	8040H	The specified path contains a logical name that represents an object that is neither a device connection nor a file connection.
E$NOUSER	8021H	The calling task's job does not have a default user, or its default user is not a user object.
E$PARAM	8004H	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. The logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system, so physical attachment is not possible.
E$PASSWORD$-MISMATCH	004BH	The password of the user object does not match the password of the corresponding user defined on the remote server.
E$PATHNAME$-SYNTAX	003EH	The specified path name contains invalid characters.
EUDFIO	02D0H	An error occurred while accessing the remote server's User Definition File (UDF). The server's UDF must have world read permission.

S$CATALOG$CONNECTION

The S$CATALOG$CONNECTION system call creates a logical name for a connection by cataloging the connection in the object directory of a specific job.

```
CALL RQ$S$CATALOG$CONNECTION(job, connection, log$name$ptr,
                             except$ptr);
```

Input Parameters

job	A TOKEN for the job in whose object directory the logical name is to be cataloged. If the value of this parameter is SELECTOR$OF(NIL), the Extended I/O System catalogs the connection in the object directory of the calling task's job.
connection	A TOKEN for the connection to be assigned the logical name. If the value of this parameter is SELECTOR$OF(NIL), the Extended I/O System obtains the connection by looking up the name in the object directory of the calling task's job.
log$name$ptr	A POINTER to a buffer containing the logical name, which must be a STRING of 12 or fewer characters. The name can be delimited with colons (:). The operating system removes the colons so that a logical name with colons is the same as one without (e.g., :F0: is effectively the same as F0); colons do not count in the length of the name. If you expect to use this logical name in other Extended I/O System calls, delimit the name with colons.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

The Extended I/O System converts the characters in the log$name$ptr STRING to uppercase and catalogs the connection in the object directory of the specified job. However, two special situations affect the outcome of this system call:

- If the job's object directory already contains the logical name, the new connection replaces the existing object in the directory. The Extended I/O System considers this to be a normal circumstance and, consequently, does not return an exception code.
- If your task sets the connection parameter to SELECTOR$OF(NIL), the Extended I/O System looks up the logical name in the object directory of the calling task's job. The system then copies the logical name and its definition into the object directory of the specified job.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONTEXT	0005H	The job in which your task is attempting to catalog the connection has an object directory that is zero bytes long.
E$EXIST	0006H	The job or connection parameter is not a token for an existing object.
E$LIMIT	0004H	At least one of the following is true: • The object directory for the specified job is already full. • The calling task's job is not an I/O job.
ELOGNAME$-NEXIST	0045H	The Extended I/O System was unable to find the specified logical name in the object directory of the calling task's job.
ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • The specified path contains a logical name that is either longer than 12 characters, has no characters, or contains invalid characters.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.

ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a connection.
E$TYPE	8002H	The job parameter is a token for an object that is not a job.

The S$CHANGE$ACCESS system call changes the access list for a named file. This system call can be used for either data or directory files.

```
CALL RQ$S$CHANGE$ACCESS(path$ptr, id, access, except$ptr);
```

Input Parameters

path$ptr — A POINTER to a STRING containing a path to the file whose access is to be changed.

id — A WORD containing the ID of the user whose access to the file is to be changed. This value can differ from the owner ID of the calling task's default user object. If the file's access list contains the ID, the Extended I/O System changes the ID's current access. If the access list does not contain the ID, the Extended I/O System adds the ID to the file's access list, unless the access list is full (contains three entries). If the access parameter described in the next paragraph is zero, this call removes the ID from the access list.

access — A BYTE defining the new access rights to be assigned to the specified user. If the entire BYTE is set to zero, the Extended I/O System removes the specified ID from the access list of the file. If the BYTE is nonzero, the meaning of the various bit settings depend upon whether the file is a data file or a directory file. The following two tables correlate the bit position and the kind of access. (System calls that start with "A$", like A$READ, are part of the Basic I/O System.)

If the bit is set to 1, access is to be granted. If the bit is set to 0, access is to be denied. (Bit 0 is low-order bit.)

DATA FILE ACCESS RIGHTS

Bit	Access
0	Delete--permission to delete the entire file by using the S$DELETE$FILE or A$DELETE$FILE system calls. Also allows changing the name of the file by using the S$RENAME$FILE or A$RENAME$FILE system call. This bit is ignored for remote files.
1	Read--permission to read data from the file by using the S$READ$MOVE or A$READ system call.
2	Append--permission to write information only at the end of the file by using the S$WRITE$MOVE or A$WRITE system call. This does not include permission to write over information already in the file or permission to truncate the file. This bit must be set to the same value as bit 3 (Update) for remote files.
3	Update--permission to write over any information in the file by using the S$WRITE$MOVE or A$WRITE system calls, and permission to truncate the file using the S$TRUNCATE$FILE or A$TRUNCATE system call. This does not include permission to add information to the end of the file. This bit must be set to the same value as bit 2 (Append) for remote files.
4-7	Reserved. Set to zero.

DIRECTORY ACCESS RIGHTS

Bits	Access
0	Delete--permission to delete the directory by using the A$DELETE$FILE or S$DELETE$FILE system call. Also allows changing the name of the directory by using the A$RENAME$FILE or S$RENAME$FILE system call. This bit is ignored for remote directories.
1	Display--permission to read information from the directory by using the A$READ, A$GET$DIRECTORY$ENTRY, or S$READ$MOVE system call.
2	Add entry--permission to add files to the directory by using the A$CREATE$FILE, A$CREATE$DIRECTORY, A$RENAME$FILE, S$CREATE$FILE, S$CREATE$DIRECTORY, or S$RENAME$FILE system call. This does not include permission to change existing entries.
3	Change entry--permission to change the access list associated with a file contained in the directory. In other words, permission to use the A$CHANGE$ACCESS or S$CHANGE$ACCESS system call. This does not include permission to add new entries or change the access list of the directory in which the file is cataloged. This bit is ignored for remote directories.
4-7	Reserved. Set to zero.

Output Parameter

except$ptr — A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

The S$CHANGE$ACCESS system call allows a task to change the access rights associated with named data or directory files. This system call can be used on any named file, including those created by the Basic I/O System.

For a task to be able to change the access rights associated with a file, the task's job must meet at least one of the following criteria:

- One of the IDs in the job's default user object is the owner of the file, or is the System Manager ID.
- One of the IDs in the job's default user object has change-entry access to the parent directory of the file.

For more information about owners, access rights, and default user objects, refer to the *iRMX® Extended I/O System User's Guide*.

Special Considerations for iRMX®-NET

You cannot change the access rights of a virtual root directory, because a virtual root directory has no assigned owner. If you attempt to change the access rights of a virtual root directory, an E$FACCESS condition code is returned.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$ALREADY$-ATTACHED	0038H	The Extended I/O System cannot attach the device containing the file because the Basic I/O System has already attached the device.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
EDEVDETACHING	0039H	The device containing the specified file is being detached.
E$DEVFD	0022H	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. In the process, it found that the device and the device driver specified in the logical attachment were incompatible.
E$FACCESS	0026H	The job containing the calling task meets none of the prerequisites for using this system call. None of the IDs in the job's default user object is the owner of the file, nor does any have change-entry access to the file's parent directory.

E$FNEXIST	0021H	One of the following conditions is true: • A file in the specified path, or the target file itself, does not exist or is marked for deletion. • The physical device was not found. The device was specified by the original call to A$PHYSICAL$ATTACH$DEVICE and is indicated in this call by the path$ptr parameter.
E$FTYPE	0027H	The specified path is attempting to use a data file as a directory.
E$IFDR	002FH	The file driver associated with this connection is the physical or stream file driver. However, the call is compatible with the named file driver only.
E$ILLVOL	002DH	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. In the process, it examined the volume label and found that the volume does not contain named files. This prevented the call from completing physical attachment because the named file driver was requested during logical attachment.
E$INVALID$FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; you should delete it.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
EIOMEM	0042H	The Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.

E$LIMIT	0004H	At least one of the following is true: • The user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*.
ELOGNAME$-NEXIST	0045H	The specified path contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • The specified path contains a logical name that is either longer than 12 characters (including colons), has no characters, or contains invalid characters.
E$MEDIA	0044H	The device containing the specified file is not on-line.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E$NAME$NEXIST	0049H	The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF).
E$NOPREFIX	8022H	You did not specify an explicit prefix (logical name), and the default prefix for the calling task's job is either undefined, or it is not a valid device connection or file connection.
ENOTFILE$CONN	0032H	In the path pointed to by path$ptr, the subpath portion is null and the prefix portion is not a file connection.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.

ENOTLOG$NAME	8040H	The specified path contains a logical name that refers to an object that is neither a device connection nor a file connection.
E$NOUSER	8021H	The calling task's job does not have a default user, or its default user is not a user object.
E$PARAM	8004H	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. The logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system. Therefore, physical attachment is not possible.
E$PASSWORD$-MISMATCH	004BH	The password of the user object does not match the password of the corresponding user defined on the remote server.
E$PATHNAME$-SYNTAX	003EH	The specified path name contains invalid characters.
E$SUPPORT	0023H	At least one of the following is true: • The calling task attempted to change access for a file other than a named file. • The calling task attempted to add another user ID to the file's access list, but the list already contains three entries. The task must delete an entry before it can add another. • The connection specified in the call is not contained in the job making the call.
EUDFIO	02D0H	An error occurred while accessing the remote server's User Definition File (UDF). The server's UDF must have world read permission.

S$CLOSE

The S$CLOSE system call closes an open connection to a named, physical, or stream file.

```
CALL RQ$S$CLOSE(connection, except$ptr);
```

Input Parameter

connection	A TOKEN for a file connection that is currently open and was opened by the S$OPEN system call.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

The S$CLOSE system call closes a connection that has been opened by the S$OPEN system call. It performs the following steps:

1. It waits until all currently running I/O operations for the file are completed.
2. It ensures that any information in a partially filled output buffer is written to the file.
3. It releases any buffers associated with the file.
4. It closes the connection to the file, deleting neither the file nor the connection.

The Extended I/O System performs no access checking before closing the connection.

The S$CLOSE system call cannot be used to close connections that were opened by the Basic I/O System. If your task attempts to do this, the Extended I/O System returns an E$CONN$NOT$OPEN exception code.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CANNOT$CLOSE	0041H	An error occurred while flushing data from EIOS buffers to an output device.

E$CONN$NOT$OPEN	0034H	One of the following conditions is true: • The connection is not open. • The connection was opened by A$OPEN rather than S$OPEN.
E$EXIST	0006H	The connection parameter is not a token for an existing object.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOMODE	0056H	One of the following is true: • A tape drive attempted to perform a read operation before the previous write operation completed. • A tape drive attempted to perform a write operation before the previous read operation completed.
EIONO$DATA	0055H	A tape drive attempted to read the next record, but it found no data.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
E$LIMIT	0004H	At least one of the following is true: • The calling task's job is not an I/O job. • The calling task's job is already involved in 255 (decimal) I/O operations.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a connection.
E$SUPPORT	0023H	The specified connection was not created by a task in the calling task's job.

S$CREATE$DIRECTORY

The S$CREATE$DIRECTORY system call creates a new directory file.

```
connection = RQ$S$CREATE$DIRECTORY(path$ptr, except$ptr);
```

Input Parameter

path$ptr	A POINTER to a STRING containing the path name of the new directory.

Output Parameters

connection	A TOKEN that represents a connection to the new directory. You can use this TOKEN as a parameter in system calls that access the directory.
except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

A task invokes this system call to create a new named-file directory. After creation, the new directory contains no entries. This system call automatically adds a new entry to the parent directory. The new directory is compatible with directories created by the Basic I/O System.

Positioning the Directory

The calling task must use the path$ptr parameter to specify the location of the new directory within the named file structure. The location indicated by the path must not be occupied. In other words, this system call can be used only to obtain connections to new, rather than existing, directories.

The default user object for the calling task's job must have add-entry access to the parent of the new directory. If the creation is successful, the first ID in the job's default user object (the owner ID) becomes the owner of the file.

The entry in the parent directory for the newly created directory provides the owner of the new directory with full access (the ability to Delete, List, Add, and Change entries) to the new directory.

Special Considerations for iRMX®-NET

You cannot create a remote directory with a virtual root directory as its parent. A virtual root directory has no owner and, thus, you cannot have write access to it. If an attempt is made to create such a remote directory, an E$FACCESS condition code is returned.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$ALREADY$-ATTACHED	0038H	The Extended I/O System cannot attach the device containing the file because the Basic I/O System has already attached the device.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
EDEVDETACHING	0039H	The device containing the specified file is in the process of being detached.
E$DEVFD	0022H	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached. In the process, it found that the device and the device driver specified in the logical attachment were incompatible.
E$FACCESS	0026H	The user object associated with the calling task's job does not have add-entry access to the parent directory.
E$FEXIST	0020H	The file already exists.
E$FNEXIST	0021H	At least one of the following is true: • A file in the specified path does not exist or is marked for deletion. • The device specified in the call is not part of the current configuration.
E$FNODE$LIMIT	003FH	The volume already contains the maximum number of files. No more fnodes are available for new files.
E$FTYPE	0027H	The specified path is attempting to use a data file as a directory.
E$ILLVOL	002DH	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached, and found that the volume does not contain named files. This prevented the call from completing physical attachment because the named file driver was requested during logical attachment.

E$INVALID$FNODE	003DH	The fnode for a directory in the specified path name is invalid. The file cannot be accessed; you should delete it.
EIOHARD	0052H	A hard I/O error occurred. This means that a retry is probably useless.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
EIOMEM	0042H	The Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.
E$LIMIT	0004H	At least one of the following is true: • The user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide.*
ELOGNAME$-NEXIST	0045H	The specified path contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • The specified path contains a logical name that is either longer than 12 characters (excluding colons), has a length of zero characters, or contains invalid characters.

E$MEDIA	0044H	The device containing the specified file is not on-line.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E$NAME$NEXIST	0049H	The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF).
E$NOPREFIX	8022H	You did not specify an explicit prefix (logical name), and the default prefix for the calling task's job is either undefined, or it is not a valid device connection or file connection.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTLOG$NAME	8040H	The specified path contains a logical name that refers to an object that is neither a device connection nor a file connection.
E$NOUSER	8021H	The calling task's job does not have a default user, or its default user is not a user object.
E$PASSWORD$-MISMATCH	004BH	The password of the user object does not match the password of the corresponding user defined on the remote server.
E$PARAM	8004H	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. The logical attachment referred to a file driver (named, physical, stream, or remote) that is not configured into your system, so physical attachment is not possible.
E$PATHNAME$-SYNTAX	003EH	The specified path name contains invalid characters.
E$SUPPORT	0023H	The NO ALLOCATE option is configured into the BIOS. You cannot create any directories on this volume.
E$SPACE	0029H	At least one of the following is true: • The volume is full. • No more files can be created on the remote server's volume. The Remote File Driver cannot distinguish between an E$FNODE$LIMIT and an E$SPACE condition code.

EUDFIO	02D0H	An error occurred while accessing the remote server's User Definition File (UDF). The server's UDF must have world read permission.

The S$CREATE$FILE system call creates a new physical, stream, or named data file. It cannot create a named directory file.

```
connection = RQ$S$CREATE$FILE(path$ptr, except$ptr);
```

Input Parameter

path$ptr	A POINTER to a STRING that contains the path name of the file to be created. The format of this path name depends on the kind of file being created. Refer to the *iRMX® Extended I/O System User's Guide* for a discussion of named, remote, physical, and stream file paths.

Output Parameters

connection	The TOKEN that represents the connection to the new file.
except$ptr	A POINTER to a WORD where the Extended I/O System returns a condition code.

Description

A task invokes this system call to create a physical, stream, or named data file, or to attach an existing file. This system call cannot be used to create or to attach a directory. (The Extended I/O System provides the S$CREATE$DIRECTORY system call for that purpose.) The file created by this system call is compatible with files created by the Basic I/O System.

If the file specified by the path$ptr parameter already exists, the Extended I/O System attempts to truncate the file to zero length and return a connection to the empty file. That is, S$CREATE$FILE acts exactly as an S$ATTACH$FILE followed by a call to S$TRUNCATE$FILE. The owner and the accessor list for the file remain unchanged.

If the file already exists, the call succeeds only if both of the following conditions are true:

- All connections to the file that are currently open allow sharing with writers.
- An ID in the default user object of the calling task's job has update access to the existing file. (This requirement applies to named files only.)

If you wish to prevent the file from being truncated accidentally, use the S$ATTACH$FILE system call; if the call to S$ATTACH$FILE returns an exception code indicating the file does not exist, you can safely use S$CREATE$FILE.

Specifying the Kind of File to be Created

The path$ptr parameter does more than simply indicate the path of the file being created. It also tells the Extended I/O System what kind of file (stream, physical, or named data) to create. The correlation between file paths and the kinds of files is discussed in detail in Chapters 4, 5, and 6 of the *iRMX® Extended I/O System User's Guide.*

Special Considerations for Named Files

These special considerations relate to named files:

- Your task must tell the Extended I/O System which directory is to be the parent of the new named file.
- To create a named file, an ID in the default user object for the calling task's job must have add-entry access to the parent directory.
- The first ID in the default user object of the calling task's job becomes the owner of the new file. The owner has full access (the owner can delete, read, append, and update the file).

Temporary Named Files

If your task invokes this system call with the path of an existing directory file, the Extended I/O System creates a temporary named data file on the device that contains the directory file. This temporary file differs from other named data files in two ways. First, the file is automatically marked for deletion, so that the Extended I/O System deletes the file as soon as your application code deletes all connections to the file. Second, the file is created without a path, so it can be accessed only through a connection.

Two access considerations apply to temporary files:

- First, any task can create a temporary file by referring to any directory. This is true because the temporary files are not listed as ordinary entries in the directory, so no add-entry access is required for the directory.
- Second, the owner of the temporary file (the first ID in the default user object of the calling task's job) has full access to the file.

Unlike local files, when you create a remote file, the remote temporary file is entered in the directory in which you are creating the remote file. Therefore, the task creating the remote file must have write access to this directory. Tasks can access this remote temporary file through its path name, as well as through connections to the file. The remote temporary file is deleted when all connections to it are deleted.

Device Considerations

Every file, regardless of kind, has an associated device. Even stream files, which have no physical devices, use the device connection to the stream file pseudo-device.

Before any file can be created, its associated device must be attached to the system.

There are two ways to attach devices to the system. One is to specify the attachment during configuration. (For more information, refer to iRMX Operating System user guides).

The second way is to attach a device while the system is running using the LOGICAL$ATTACH$DEVICE system call.

Special Considerations for iRMX®-NET

You cannot create a remote file with a virtual root directory as its parent. A virtual root directory has no owner and, thus, you cannot have write access to it. If an attempt is made to create such a remote file, an E$FACCESS condition code is returned.

Codes

E$OK	0000H	No exceptional conditions.
E$ALREADY$-ATTACHED	0038H	The Extended I/O System cannot attach the device containing the file because the Basic I/O System has already attached the device.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
EDEVDETACHING	0039H	The device containing the specified file is being detached.
E$DEVFD	0022H	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached, and found that the device and the device driver specified in the logical attachment were incompatible.
E$FACCESS	0026H	At least one of the following is true: • The default user object associated with the calling task's job does not have add-entry access to the parent directory. • The default user object associated with the calling task's job does not have update access to the existing file with the specified path name.

E$FNEXIST	0021H	At least one of the following is true: • A file in the specified path does not exist or is marked for deletion. • The physical device specified in the call was not found.
E$FNODE$LIMIT	003FH	The volume already contains the maximum number of files. No more fnodes are available for new files.
E$FTYPE	0027H	The specified path is attempting to use a data file as a directory.
E$ILLVOL	002DH	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached, and found that the volume does not contain named files. This prevented the call from completing physical attachment.
E$INVALID$FNODE	003DH	The fnode for a directory in the specified path name is invalid. The file cannot be accessed; you should delete it.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
EIOMEM	0042H	The Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.

E$LIMIT	0004H	At least one of the following is true: • The calling task has reached the object's limit. • The user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*.
ELOGNAME$-NEXIST	0045H	The specified path contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • The specified path contains a logical name that is either longer than 12 characters (including colons), does not contain at least one character, or contains invalid characters.
E$MEDIA	0044H	The device containing the specified file is not on-line. The media maybe inserted incorrectly (upside down).
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E$NAME$NEXIST	0049H	The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF).
E$NOPREFIX	8022H	You did not specify an explicit prefix (logical name), and the default prefix for the calling task's job is either undefined, or it is not a valid device connection or file connection.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTLOG$NAME	8040H	The specified path contains a logical name that refers to an object that is neither a device connection nor a file connection.

E$NOUSER	8021H	The calling task's job does not have a default user object, or the object cataloged in R?IOUSER is not a user object.
E$PASSWORD$-MISMATCH	004BH	The password of the user object does not match the password of the corresponding user defined on the remote server.
E$PARAM	8004H	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached. The logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system, so the physical attachment is not possible.
E$PATHNAME$-SYNTAX	003EH	The specified path name contains invalid characters.
E$SHARE	0028H	You are trying to create a file that already exists. The Extended I/O System must truncate the existing file to zero length to do the create. This truncate to zero length failed for one or more of the following reasons: • Another open connection does not allow sharing with writers. • The default user for the calling task's job does not have update access to the file.
E$SPACE	0029H	At least one of the following is true: • The volume is full. • No more files can be created on the remote server's volume. The Remote File Driver cannot distinguish between an E$FNODE$LIMIT and an E$SPACE condition code.
E$SUPPORT	0023H	One of the following is true: • The NO CREATE FALSE option is configured into the BIOS. • The NO TRUNCATE option is configured into the BIOS.
EUDFIO	02D0H	An error occurred while accessing the remote server's User Definition File (UDF). The server's UDF must have world read permission.

The S$DELETE$CONNECTION system call deletes a file connection. It cannot delete a device connection.

```
CALL RQ$S$DELETE$CONNECTION(connection, except$ptr);
```

Input Parameter

connection	A TOKEN for the file connection to be deleted.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

This system call deletes a file connection, but it cannot delete a device connection. If the connection is open, the S$DELETE$CONNECTION system call automatically closes it before deleting it.

If the file has been marked for deletion (by a previous system call) and there are no more connections to the file, then S$DELETE$CONNECTION deletes the file.

The Extended I/O System does not check access before deleting a connection.

The S$DELETE$CONNECTION system call can be used with connections that were created by the Basic I/O System as long as the connections meet the requirements discussed in the *iRMX® Extended I/O System User's Guide*, Appendix E.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$EXIST	0006H	The connection parameter is not a token for an existing object.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.

EIOMODE	0056H	One of the following is true: • A tape drive attempted to perform a read operation before the previous write operation completed. • A tape drive attempted to perform a write operation before the previous read operation completed.
EIONO$DATA	0055H	A tape drive attempted to read the next record, but it found no data.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
E$LIMIT	0004H	At least one of the following is true: • The associated job or the job's default user object is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a file connection.
E$SUPPORT	0023H	The specified connection was not created by a task in this job.

The S$DELETE$FILE system call deletes a stream, named data, or named directory file. This system call cannot delete a physical file.

```
CALL RQ$S$DELETE$FILE(path$ptr, except$ptr);
```

Input Parameter

path$ptr — A POINTER to a STRING that specifies the path for the file to be deleted. The form of the path depends upon the kind of file. (See the *iRMX® Basic I/O System User's Guide* for information on path syntax.)

Output Parameter

except$ptr — A POINTER to a WORD where the Extended I/O System returns a condition code.

Description

A task can use this system call whenever the task needs to delete a stream, named data, or named directory file. This system call marks the specified file for deletion, but the Extended I/O System postpones deletion until the following conditions are met:

- For stream and named data files, there is only one condition. The deletion occurs as soon as no connections to the file remain. Your tasks can use the S$DELETE$CONNECTION system call to delete connections.
- For named directories there are two conditions. The directory must be empty, and no connections to the directory can remain. The Extended I/O System returns an EDIRNOT$EMPTY condition code if a task attempts to delete a non-empty directory. This system call can delete files created by the Basic I/O System as well as those created by the Extended I/O System. Refer to the *iRMX® Extended I/O System User's Guide*, Appendix E, for a general discussion of compatibility between the Extended and Basic I/O Systems.

If the task attempts to delete a named data or directory file, the default user object of the task's job must have deletion access to the file.

Special Considerations for iRMX®-NET

You cannot delete a remote file which has a virtual root directory as its parent, because a virtual root directory has no assigned owner. To delete a file, you must have write access to its parent directory. If you attempt to delete a remote file whose parent directory is a virtual root directory, an E$FACCESS condition code is returned.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$ALREADY$-ATTACHED	0038H	The specified device is already attached.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
EDEVDETACHING	0039H	The device containing the specified file is in the process of being detached.
E$DEVFD	0022H	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached, and found that the device and the device driver specified in the logical attachment were incompatible.
EDIRNOT$EMPTY	0031H	Your task is attempting to delete a directory that is not empty.
E$FACCESS	0026H	At least one of the following is true: • The default user object associated with the calling task's job does not have delete access to the specified file. • The call is attempting to delete a bit map file or the root directory.
E$FNEXIST	0021H	At least one of the following is true: • A file in the specified path, or the target file itself, does not exist or is marked for deletion. • The physical device was not found. The device was specified by the original call to A$PHYSICAL$ATTACH$DEVICE and is indicated in this call by the path$ptr parameter.
E$FTYPE	0027H	The specified path contains a file name that should be the name of a directory, but is not. (Except for the last file, each file in a path must be a directory.)

E$ILLVOL	002DH	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached, and found that the volume does not contain named files. This prevented the call from completing physical attachment because the named file driver was requested during logical attachment.
E$IFDR	002FH	The specified file is a physical file.
E$INVALID$FNODE	003DH	The fnode associated with a file is either marked not allocated, or the fnode number is out of range.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
EIOMEM	0042H	The Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.
E$LIMIT	0004H	At least one of the following is true: • Either the user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job. • Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*.
ELOGNAME$-NEXIST	0045H	The specified path contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, global job, or the root job.

ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • The specified path contains a logical name that is either longer than 12 characters (including colons), contains no characters, or contains invalid characters.
E$MEDIA	0044H	The device containing the specified file is off-line.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E$NAME$NEXIST	0049H	The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF).
E$NOPREFIX	8022H	You did not specify an explicit prefix (logical name), and the default prefix for the calling task's job is either undefined, or it is not a valid device connection or file connection.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTFILE$CONN	0032H	In the path pointed to by path$ptr, the subpath portion is null and the prefix portion is not a file connection.
ENOTLOG$NAME	8040H	The specified path contains a logical name that refers to an object that is neither a device connection nor a file connection.
E$NOUSER	8021H	The calling task's job does not have a default user object, or the object cataloged in R?IOUSER is not a user object.
E$PASSWORD$-MISMATCH	004BH	The password of the user object does not match the password of the corresponding user defined on the remote server.
E$PARAM	8004H	The Extended I/O System attempted to physically attach a device that is logically attached. That logical attachment refers to a file driver (named, physical, or stream) that is not configured into your system. Therefore, physical attachment is not possible.

E$PATHNAME$-SYNTAX	003EH	The specified path name contains invalid characters.
E$SUPPORT	0023H	The task is attempting to delete a physical file.
EUDFIO	02D0H	An error occurred while accessing the remote server's User Definition File (UDF). The server's UDF must have world read permission.

SGETCONNECTION$STATUS

The SGETCONNECTION$STATUS system call provides status information about file and device connections.

```
CALL RQ$S$GET$CONNECTION$STATUS(connection, info$ptr, except$ptr);
```

Input Parameter

connection — A TOKEN for the connection whose status is desired.

Output Parameters

info$ptr — A POINTER to a structure in which the Extended I/O System places the status information. You can provide the memory for this structure by requesting an iRMX segment, or by reserving it in your code. The structure must have the following form:

```
DECLARE connection$info STRUCTURE(
                file$driver         BYTE,
                flags               BYTE,
                open$mode           BYTE,
                share$mode          BYTE,
                file$pointer        DWORD,
                access              BYTE,
                number$buffers      BYTE,
                buffer$size         WORD,
                seek                BOOLEAN)
```

where

file$driver — Identifies the kind of file associated with the connection.

1 physical file
2 stream file
4 named file
5 remote

flags — Indicates the kind of connection this is. If Bit 1 is one, the connection is capable of being opened. If Bit 2 is one, the connection is a device connection. (Bit zero is the low-order bit.)

open$mode — Indicates the purpose for which the connection was opened. This applies only to file connections.

0	closed
1	open for reading only
2	open for writing only
3	open for both reading and writing

share$mode — Indicates who can share the connection. Applies to both device and file connections.

0	cannot be shared
1	share with readers only
2	share with writers only
3	share with anybody

file$pointer — A 32-bit offset from the beginning of the file where the next I/O operation will be performed.

access — The access rights that were computed when the connection was created. (The access rights for remote files are computed during operations on the connection, not at creation time.) This information applies only to connections for named files, and the interpretation of the information depends upon whether the file is a data file or a directory. Access is represented as a bit mask. In the following tables, access is granted if a bit is set to one (bit zero is the low-order bit.).

Bit	Data File	Directory
0	Delete	Delete
1	Read	List
2	Append	Add Entry
3	Update	Change Entry
4-7	Reserved	Reserved

For remote files, the access bits are interpreted as follows:

Bit	Data File	Directory
0	Ignored	Ignored
1	Read	Display
2	Write (must be set the same as bit 3)	Write
3	Write (must be set the same as bit 2)	Ignored
4-7	Reserved	Reserved

number$buffers — The number of buffers used with this connection. This applies only to file connections.

buffer$size — The size, in bytes, of each buffer used with the connection.

seek — Tells whether or not the SEEK function can be used with this connection. Zero means no, and 0FFh means yes.

except$ptr — A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

The SGETCONNECTION$STATUS system call allows a task to obtain status information about file connections and device connections that were created by either the Basic I/O System or the Extended I/O System. The nature of the returned information depends upon whether the connection is for a file or a device. Some of the information also depends on the kind of file associated with the connection.

The Extended I/O System does not check access before returning status information.

Although you can use this system call with connections created by the Basic I/O System, you must adhere to the restrictions described in the *iRMX® Extended I/O System User's Guide*, Appendix E.

Special Considerations for iRMX®-NET

When the status of a file connection to a virtual root directory is requested, display permission is granted and write permission is denied. As a result, bit 1 of the access field is set to 1 and bit 2 is set to 0.

Also, unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONN$NOT$OPEN	0034H	The connection was opened by the A$OPEN system call rather than the S$OPEN system call.
E$EXIST	0006H	The connection parameter is not a token for an existing job.
E$IFDR	002FH	An invalid file driver request occurred.
E$LIMIT	0004H	At least one of the following is true: • The calling task has reached its object limit. • Either the calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a connection.
ENOTFILE$CONN	0032H	For remote files, the connection parameter must be a file connection, not a device connection.
E$SUPPORT	0023H	The specified connection was not created by a task in the calling task's job.

The SGETDIRECTORY$ENTRY system call returns a directory entry name to the caller. A directory entry name is a single path component for a file whose parent is the directory.

> **iRMX I Note:** The SGETDIRECTORY$ENTRY system call is not supported in the iRMX I Operating System.

```
CALL RQ$S$GET$DIRECTORY$ENTRY(dir$name$ptr, entry$num, name$ptr,
                              except$pt);
```

Input Parameters

dir$name$ptr	A POINTER to a STRING containing the directory path name. This path name can be up to 255 characters long.
entry$num	A WORD giving the entry number of the desired file name. Entries in a directory are numbered sequentially starting from zero. The E$EMPTY$ENTRY condition code will be returned if there is no directory entry associated with the number.

Output Parameter

name$ptr	A POINTER to a buffer where the system will return the entry name. This name, a maximum length of 14 BYTES, corresponds to the entry number given by the user in the entry$num parameter.
except$ptr	A POINTER to a WORD where the condition code will be returned.

Description

The SGETDIRECTORY$ENTRY system call applies to named files only. When called, it returns the file name associated with a specified directory entry. This name is a single subpath component for a file whose parent is the designated directory. As an alternative to using this system call, an application task can open and read a directory file.

NOTE

The caller must have List access to the designated directory.

Special Considerations for iRMX®-NET

The SGETDIRECTORY$ENTRY system call is not supported for remote directories. However, remote directories can be read with the A$OPEN, A$READ, S$OPEN, and S$READ$MOVE system calls.

Condition Codes

E$OK	0000H	No exceptional conditions.
EDIREND	0025H	The entry$num parameter is greater than the number of entries in the directory.
E$EMPTY$ENTRY	0024H	The file entry designated in the call is empty.
E$FACCESS	0026H	The caller's default user object is not qualified for list access to the directory.
E$FTYPE	0027H	The specified connection does not refer to a directory.
E$IFDR	002FH	One of the following is true: • This system call applies only to named directories, but the STRING pointed to by dir$name$ptr specifies another type of file. • This system call is not supported for remote files.
E$IO	002BH	An I/O error occurred that might have prevented the operation from completing.
E$LIMIT	0004H	The calling task's job has already reached its object limit.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete this call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.

The SGETFILE$STATUS system call allows a task to obtain information about a physical, stream, or named file.

```
CALL RQ$S$GET$FILE$STATUS(path$ptr, info$ptr, except$ptr);
```

Input Parameter

path$ptr — A POINTER to a STRING that contains the path for the file. The format of this path varies from one kind of file to another. Refer to the *iRMX® Extended I/O System User's Guide* for path syntax.

Output Parameters

info$ptr — A POINTER to a structure where the Extended I/O System returns the status information. You must allocate this memory, either in your program code space or as an iRMX segment. The structure has the form described here.

The information in the first part of this structure--down to the device$connections field--is returned for any file (physical, stream, or named), but information from the file$id field to the end of the structure is present only for named files. The contents of the named$file field indicate whether the file is a named file.

```
DECLARE file$info STRUCTURE(
            device$share            WORD,
            number$connections      WORD,
            number$reader           WORD,
            number$writer           WORD,
            share                   BYTE,
            named$file              BYTE,
            device$name(14)         BYTE,
            file$drivers            WORD,
            functions               BYTE,
            flags                   BYTE,
            device$granularity      WORD,
            device$size             DWORD,
            device$connections      WORD,
```

Information from this point on is returned only if the file is a named file.

```
file$id              WORD,
file$type            BYTE,
file$granularity     BYTE,
owner$id             WORD,
create$time          DWORD,
access$time          DWORD,
modify$time          DWORD
file$size            DWORD,
file$blocks          DWORD,
volume$name(6)       BYTE,
volume$granularity   WORD,
volume$size          DWORD,
accessor$count       WORD,
owner$access         BYTE);
```

The meanings of these fields are:

device$share — Indicates whether or not the device can be shared. Currently, this word is always set to 1, indicating that all devices can be shared.

number$connections — The number of connections to the file.

For remote files, this field indicates the number of connections the calling job has to the file.

number$reader — The number of connections currently open for reading.

For remote files, this field is set as follows:

Connection	number$reader
No connection	0
Connection open - read	1
Connection open - write	0
Connection open - read/write	1

number$writer — The number of connections currently open for writing.

For remote files, this field is set as follows:

Connection	number$writer
No connection	0
Connection open - read	0
Connection open - write	1
Connection open - read/write	1

share

The current shared status of the file; possible values are

0 Private use only
1 Share with readers only
2 Share with writers only
3 Share with all users

For remote files, a value of 3 is returned if the specified remote file connection is not open. If the remote file is open, the share mode used to open the connection is returned.

named$file

Tells whether this structure contains any information beyond the device$connections field. 0FFh means yes and 0 means no. 0FFh is always returned for remote files.

device$name

The name of the physical device where this file resides. This name is padded with blanks. To ensure the uniqueness of device names, they should not be more than 14 characters in length.

For remote files, the name of the remote server on which the file resides is returned.

file$drivers

A bit map that tells what kinds of files can reside on this device. If bit n is on, then file driver n+1 can be used. Bit 0 is the low-order bit.

Bit	Driver No.	Driver
0	1	Physical file
1	2	Stream file
2	3	Reserved
3	4	Named file
4	5	Remote file

functions

A bit map that describes the functions supported by the device where this file resides. A bit set to one indicates the corresponding function is supported. Bit 0 is the low-order bit.

This field is not supported by iRMX-NET. A value of 0 is always returned for remote files.

Bit	Function
0	F$READ
1	F$WRITE
2	F$SEEK
3	F$SPECIAL
4	F$ATTACH$DEV
5	F$DETACH$DEV
7	F$CLOSE

flags

Meaningful only for diskette drives. This field is interpreted as follows. (Bit 0 is the low-order bit.)

This field is not supported by iRMX-NET. A value of 0 is always returned for remote files.

Bit	Meaning
0	0=bits 1-7 not significant 1=bits 1-7 are significant
1	0=single density 1=double density
2	0=single sided 1=double sided
3	0=8-inch diskette 1=5 1/4-inch diskette
4	0=standard diskette, meaning that track 0 is single-density with 128 byte sectors 1=a nonstandard diskette or not a diskette
5-7	reserved

device$granularity

The granularity, in bytes, of the device where this file resides.

device$size

The storage capacity of the device, in bytes.

device$connections

The number of connections to the device.

For remote files, this field contains the number of connections that local users have to files on the remote server.

The information from here to the end of the structure is returned only for named files, as indicated by a value of 0FFh in the named$file field.

file$id	A number that distinguishes this file from all other files on the same device.
file$type	The file type: 6 means directory file and 8 means data file.
file$granularity	The file granularity, as a multiple of volume$granularity. For example, if file$granularity is 2 and volume$granularity is 256, then the file's granularity is 512. A value of 1 is always returned for remote files.
owner$id	The first ID in the creating task's default user object.
create$time	The time and date when the file was created. Whether the operating system maintains this field is a configuration option.
access$time	The time and date when the file was last accessed. Whether the operating system maintains this field is a configuration option.
modify$time	The time and date when the file was last modified. Whether the operating system maintains this field is a configuration option.
file$size	The total size of the file, in bytes.
file$blocks	The number of volume blocks allocated to this file. A volume block is a contiguous area of storage that contains volume$granularity bytes of data.
volume$name	The left-adjusted, null-padded ASCII name for the volume containing this file.
volume$granularity	The volume granularity, in bytes.
volume$size	The storage capacity, in bytes, of the volume on which this file is stored.
accessor$count	The number of IDs in the file's accessor list.

owner$access — The access rights to this file that are currently held by the owner. The access rights are encoded in a bit mask that you can interpret by using the following table. Remember that Bit 0 is the low-order bit, and that access is granted if the corresponding bit is set to 1.

Bit	Data File	Directory
0	Delete	Delete
1	Read	List
2	Append	Add Entry
3	Update	Change Entry
4-7	Reserved	Reserved

except$ptr — A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

This system call provides the calling task with information about the status of a file. Fields through the device$connections field are always returned if the call is successful. Fields following the device$connections field are returned only when the file being referred to is a named file, as indicated by the named$file field being 0FFh.

The Extended I/O System does not check access before returning file status information.

This system call can be used with any file, including those created by the Basic I/O System. However, because of the asynchronous nature of some of the Basic I/O System calls, there is some chance that the information returned might be inaccurate. For instance, if your application code invokes the SGETFILE$STATUS system call while the Basic I/O System is processing an A$WRITE for the same file, the values returned in the file size fields might be incorrect. Refer to the *iRMX® Extended I/O System User's Guide*, Appendix E for a more general discussion of compatibility between the Extended and Basic I/O Systems.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$ALREADY$-ATTACHED	0038H	The Extended I/O System is unable to attach the device containing the file because the Basic I/O System has already attached the device.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
EDEVDETACHING	0039H	The device containing the specified file is in the process of being detached.

E$DEVFD	0022H	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. In the process, it found that the device and the device driver specified in the logical attachment were incompatible.
E$FNEXIST	0021H	At least one of the following is true: • A file in the specified path, or the target file itself, does not exist or is marked for deletion. • The physical device specified in the call was not found.
E$FTYPE	0027H	The specified path contains a file name that should be the name of a directory, but is not. (Except for the last file, each file in a path must be a directory.)
E$ILLVOL	002DH	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. In the process, it examined the volume label and found that the volume does not contain named files. This prevented the Extended I/O System from completing physical attachment because the named file driver was requested during logical attachment.
E$INVALID$FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; you should delete it.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOMODE	0056H	One of the following is true: • A tape drive attempted to perform a read operation before the previous write operation completed. • A tape drive attempted to perform a write operation before the previous read operation completed.
EIONO$DATA	0055H	A tape drive attempted to read the next record, but it found no data.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.

EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOMEM	0042H	The Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.
E$LIMIT	0004H	At least one of the following is true: • The user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job. • The calling task's object limit has been reached.
ELOGNAME$-NEXIST	0045H	The specified path contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • contains a logical name that is either longer than 12 characters (including colons), has no characters, or contains invalid characters.
E$MEDIA	0044H	The device containing the specified file is not on-line.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E$NOPREFIX	8022H	You did not specify an explicit prefix (logical name), and the default prefix for the calling task's job is either undefined, or it is not a valid device connection or file connection.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTFILE$CONN	0032H	For remote files, the connection parameter must be a file connection, not a device connection.

ENOTLOG$NAME	8040H	The specified path contains a logical name that refers to an object that is neither a device connection nor a file connection.
E$NOUSER	8021H	The calling task's job does not have a default user, or its default user is not a user object.
E$PARAM	8004H	The Extended I/O System attempted the physical attachment of a device that had formerly been only logically attached. In the process, it found that the logical attachment referred to a file driver (named, physical, or stream) that is not configured into your system. Therefore the physical attachment is not possible.
E$PATHNAME$-SYNTAX	003EH	The specified path name contains invalid characters.
EUDFIO	02D0H	An error occurred while accessing the remote server's User Definition File (UDF). The server's UDF must have world read permission.

SGETPATH$COMPONENT returns the name of a named file as the file is known in its parent directory.

> **iRMX I Note:** The SGETPATH$COMPONENT system call is not supported in the iRMX I Operating System.

```
CALL RQ$S$GET$PATH$COMPONENT(connection, name$ptr, except$ptr);
```

Input Parameters

connection	A TOKEN for the file connection whose name is desired.

Output Parameter

name$ptr	A POINTER to a STRING where the system returns the path component. The maximum length of the STRING is 14 BYTES.
except$ptr	A POINTER to a WORD where the Extended I/O System returns condition codes.

Description

The format of the component returned by this call is dependent on the type of file driver employed by the call. A null string is returned under the following circumstances:

- If the file driver is Named or Remote and the connection is to the root directory of a volume.
- If the file driver's connection accesses either Stream or Physical files.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONTEXT	0005H	The name$ptr parameter is equal to NIL.
E$FNEXIST	0021H	The file is marked for deletion. (In this case, the string is undefined.)
E$INVALID$FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; you should delete it.

E$IO	002BH	An I/O error occurred that might have prevented the operation from completing.
EIOMEM	0042H	The memory available to the EIOS is not sufficient to complete the call.
ENOTFILE$CONN	0032H	For remote files, the connection parameter must be a file connection, not a device connection.

The S$LOOK$UP$CONNECTION system call accepts a logical name from the calling task and returns a token for the connection associated with the logical name.

```
connection = RQ$S$LOOK$UP$CONNECTION(log$name$ptr, except$ptr);
```

Input Parameter

log$name$ptr	A POINTER to a STRING (of 1 to 12 characters) containing the logical name to be looked up. The name can be delimited with colons (:). The operating system removes the colons so that a logical name with colons is the same as one without (e.g., :F0: is effectively the same as F0). Colons do not count in the length of the name.

Output Parameters

connection	The TOKEN that represents the connection associated with the logical name.
except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

After converting any lowercase letters in the logical name to uppercase, the Extended I/O System searches for the logical name. It first checks the object directory of the local job, then the global job, and finally the root job. (This progressively more global search sequence is described more completely in the iRMX\ Extended I/O System User's Guide.) When it finds the logical name, the Extended I/O System returns the token for the connection.

Your tasks can invoke this system call to look up logical names created by the Nucleus system call CATALOG$OBJECT. However, CATALOG$OBJECT does not convert from lowercase to uppercase. So if you desire compatibility, use uppercase characters when you use the CATALOG$OBJECT system call.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONTEXT	0005H	The calling task's job is not an I/O job.

E$LIMIT	0004H	The calling task's job is not an I/O job.
ELOGNAME$-NEXIST	0045H	The specified path contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
ELOGNAME$-SYNTAX	0040H	The specified logical name contains at least one of the following syntax errors: • The specified path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • The specified path contains a logical name that is either longer than 12 characters, has no characters, or contains invalid characters.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The logical name refers to an object that is not a connection.
E$TIME	0001H	The calling task's job is not an I/O job.

The S$OPEN system call opens a file connection so that your tasks can access the file.

```
CALL RQ$S$OPEN(connection, mode, number$buffers, except$ptr);
```

Input Parameters

connection — A TOKEN for the file connection to be opened. The connection must have been created in the calling task's job. If the connection was created in a different job, use S$ATTACH$FILE to obtain a new connection.

mode — A BYTE telling how your task is going to use the connection and with whom it will share the connection. You should set the BYTE as follows:

Value	How Connection is Used
1H	For reading only; share with all.
2H	For writing only; share with all.
3H	For both reading and writing; share with all.
4H	For reading only; private use.
5H	For writing only; private use.
6H	For both reading and writing; private use.
7H	For reading only; share with readers.
8H	For writing only; share with readers.
9H	For both reading and writing; share with readers.
0AH	For reading only; share with writers.
0BH	For writing only; share with writers.
0CH	For both reading and writing; share with writers.

Remote directories must be opened with the mode parameter set to 1.

number$buffers — A BYTE containing the number of buffers that you want the Extended I/O System to allocate for this connection. This number must be between zero and a maximum value that you specified when you configured the Basic I/O System.

Output Parameter

except$ptr — A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

This system call performs the following functions:

- It creates the number of buffers requested.
- It sets the connection's file pointer to NIL.
- It starts reading ahead if the number of buffers is greater than zero and the mode parameter includes reading.

Access Rights and Selecting a Mode

When you specify the mode, you must be accurate or err on the side of generosity. If you are not certain how the connection will be used, specify both reading and writing.

In the case of named files, the mode that you specify must match the access rights of the connection. (These are the access rights that the Extended I/O System assigned the connection when the connection was created.) For example, if your task attempts to open for reading a connection that has access for writing only, the Extended I/O System returns an E$FACCESS exception code.

Selecting the Number of Buffers

Deciding how many buffers to allocate for file I/O is based on two considerations--memory and performance. The amount of memory used for buffers is directly proportional to the number of buffers. So you can save memory by using fewer buffers.

The performance consideration is more complex. Up to a certain point, the more buffers you allocate, the faster your task can run. The actual break-even point, the point where more buffers don't improve performance, depends on many variables. Be aware that in order to overlap I/O with computation, you must specify at least two buffers.

If performance is important, and you have no idea how many buffers to specify, start with two. Once your task is running successfully, you can experiment, adding or removing buffers until you have found the optimum number of buffers.

If your application performs a great deal of random access file I/O (for example, many seek/read or seek/write combinations), performance may be enhanced by specifying zero buffers.

If performance is not so important and memory is, use zero buffers.

Special Considerations for iRMX®-NET

Unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONN$OPEN	0035H	The connection is already open.
EDEVOFF$LINE	002EH	The device being accessed is now offline.
E$EXIST	0006H	The connection parameter is not a token for an existing object.
E$FACCESS	0026H	The access rights embedded in the connection prohibit opening the file in the specified mode.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOMODE	0056H	One of the following is true: • A tape drive attempted to perform a read operation before the previous write operation completed. • A tape drive attempted to perform a write operation before the previous read operation completed.
EIONO$DATA	0055H	A tape drive attempted to read the next record, but it found no data.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.

E$LIMIT	0004H	At least one of the following is true: • The calling task's job is not an I/O job. • The calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations. • Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a file connection.
ENOTFILE$CONN	0032H	The connection is a device connection.
E$PARAM	8004H	The mode parameter is set to a value other than 1 through C hexadecimal.
E$SHARE	0028H	At least one of the following is true: • The call attempted to open a directory file or a bit-map file for writing. • The file's sharing attribute is currently not compatible with the mode specified in this call. • The call attempted to open a remote directory with the mode parameter set to a value other than 1.
E$SUPPORT	0023H	The specified connection was not created by a task in the calling task's job.

The S$READ$MOVE reads a number of bytes from a file to a buffer.

```
bytes$read = RQ$S$READ$MOVE(connection, buffer$ptr, bytes$desired,
                            except$ptr);
```

Input Parameters

connection	A TOKEN for the connection to the file. This connection must be open for reading or for both reading and writing, and the file pointer of the connection must point to the first byte to be read.
bytes$desired	A WORD containing the maximum number of bytes you want to read from the file.

Output Parameters

bytes$read	A WORD containing the actual number of bytes that the Extended I/O System reads from the file.
buffer$ptr	A POINTER to a buffer that will receive the information that the Extended I/O System reads from the file.
except$ptr	A POINTER to a WORD where the Extended I/O System returns a condition code.

Description

This system call reads a collection of contiguous bytes from the file associated with the connection. These bytes are placed in a buffer specified by the calling task.

Creating the Buffer

The buffer$ptr parameter tells the Extended I/O System where to place the bytes after they are read. You must create this buffer because the Extended I/O System does not. To create the buffer, make an iRMX segment, or create a buffer during the compilation of your program. You must ensure that the buffer is long enough.

In the iRMX II Operating System, if you use an iRMX segment as your buffer, the 80286 microprocessor's built-in abilities will detect when a task attempts to write beyond a buffer. If you create a buffer at compilation time, the Extended I/O System will not sense when overwriting occurs. If your task attempts to read more bytes than the buffer is capable of holding, the information immediately following the buffer could be overwritten.

> **iRMX I Note:** The iRMX I Operating System does not run in protected mode on the 80286 microprocessor. Therefore, the iRMX I Operating System cannot detect when a task attempts to write beyond the end of a buffer.

The number of bytes that your task requests (bytes$desired) is the maximum number of bytes that the Extended I/O System places in the buffer. However, there are two circumstances under which the Extended I/O System reads fewer bytes.

- First, if the Extended I/O System detects an end-of-file before reading the number of bytes requested, it returns only those bytes preceding the end-of-file. In this case, the bytes$read parameter can be less than the bytes$desired parameter without generating an exceptional condition.
- Second, if an exceptional condition occurs during the reading operation. In this case, the information in the buffer and the value of the bytes$read parameter are meaningless.

If your task performs random-access reads of the file, it must identify which bytes to read from the file by using the S$SEEK system call to position the connection's file pointer to the first byte that it wants to read.

In contrast, if your task reads from the file sequentially, the Extended I/O System maintains the connection's file pointer automatically.

Effects of Priority

The priority of the task invoking this system call can greatly affect the performance of the application system. For better performance, the priority of the invoking task should be equal to or lower than (numerically greater than) 130. If the priority of the calling task is greater than 130, the operating system cannot overlap the read operation with computation or with other I/O operations. (To find out how to set priorities for application tasks, refer to the *iRMX® I Nucleus User's Guide* or the *iRMX® II Nucleus User's Guide.*)

Special Considerations for iRMX®-NET

iRMX-NET's remote file driver does not perform fragmentation and reassembly. For optimal performance, reading and writing should begin at offsets that are integral multiples of the remote server's buffer size. The device$granularity parameter returned by the SGETFILE$STATUS system call indicates the buffer size of a remote server.

Condition Codes

E$OK	0000H	No exceptional conditions.
EBADBUFF	8023H	**This condition code is not supported in the iRMX I Operating System.** One of the following is true: • The specified memory buffer is not writeable. • The specified memory buffer crosses a segment boundary.
E$CONN$NOT$OPEN	0034H	At least one of the following is true: • The connection is not open for reading or for both reading and writing. • The connection is closed. • The connection was opened by the A$OPEN system call rather than the S$OPEN system call.
E$EXIST	0006H	The connection is not a token for an existing object.
E$FLUSHING	002CH	The specified device is being detached.
E$IDDR	002AH	This request is invalid for the device driver. For example, it is not valid to use this call with a line printer.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOMODE	0056H	One of the following is true: • A tape drive attempted to perform a read operation before the previous write operation completed. • A tape drive attempted to perform a write operation before the previous read operation completed.
EIONO$DATA	0055H	A tape drive attempted to read the next record, but it found no data.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.

EIOUNCLASS	0050H	An unknown type of I/O error occurred.
E$LIMIT	0004H	At least one of the following is true: • The calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCON-FIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a file connection.
E$SPACE	0029H	At least one of the following is true: • This call attempted to read beyond the end of the volume. • Another task is writing to the file using the same connection and is attempting to write beyond the end of the volume or the end of the available space on the volume.
E$SUPPORT	0023H	The connection parameter was not created by a task in the calling task's job.

The S$RENAME$FILE system call changes the name of a directory or data file. It cannot be used for stream or physical files.

```
CALL RQ$S$RENAME$FILE(path$ptr, new$path$ptr, except$ptr);
```

Input Parameters

path$ptr	A POINTER to a STRING that specifies the current path for an existing file that is to be renamed. The syntax of this path is described in Chapter 4 of the *iRMX® Extended I/O System User's Guide*.
new$path$ptr	A POINTER to a STRING that specifies the new path for the file. This path must comply with the syntax and semantics of paths for named files as discussed in Chapter 4 of the *iRMX® Extended I/O System User's Guide*. Furthermore, this path cannot refer to an existing file.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns a condition code.

Description

This system call, which can be used only with named files, allows your task to change the path for a file. You can rename directory files as well as data files.

NOTE

When you rename a directory, you change the paths for all files and other directories contained in the directory.

Restrictions

If your task is renaming a file, the task can change any aspect of the file's path so long as the file remains on the same volume. If you are renaming a directory, it must still have the same parent directory (the directory above the one being renamed).

To be able to rename a file, the default user object of the calling task's job must have two kinds of access:

- Deletion access to the original file
- Add-entry access to the file's new parent directory

Special Considerations for iRMX®-NET

The S$RENAME$FILE system call cannot rename the following files and directories on a remote server:

- a file in a virtual root directory
- a virtual root directory
- a public directory

If an attempt is made to rename any of these files and directories, an E$FACCESS exceptional condition is returned.

Also, unlike a local named file, the access rights of a remote named file are not checked when a connection to the file is created. Instead, the remote named file's access rights are checked during operations on the connection.

The above discrepancy won't affect your programs if you do the following:

- Open, delete, and rename files prior to changing their access lists.
- Establish connections to files after changing their access lists.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$ALREADY$-ATTACHED	0038H	The Extended I/O System is unable to attach the device containing the file because the Basic I/O System has already attached the device.
E$CONTEXT	0005H	The calling task's job is not an I/O job.
EDEVDETACHING	0039H	The device containing the specified file is in the process of being detached.

E$DEVFD	0022H	The Extended I/O System attempted to physically attach a device that had been only logically attached, and found that the device and the device driver specified in the logical attachment were incompatible.
E$FACCESS	0026H	At least one of the following is true: • The call is trying to rename a bit-map file or the root directory. • The default user object associated with the calling task's job does not have add-entry access to the parent directory of the new$path$ptr file. • The default user object associated with the calling task's job does not have delete access to the file to be renamed.
E$FEXIST	0020H	The new$path$ptr parameter refers to a file that already exists.
E$FNEXIST	0021H	A file in the specified path, or the file being renamed, does not exist or is marked for deletion.
E$FTYPE	0027H	The specified path contains a file name that should be the name of a directory, but is not. (Except for the last file, each file in a path must be a directory.)
E$IFDR	002FH	The specified file is a stream or physical file.
E$ILLOGICAL$-RENAME	003BH	The call attempted to rename a directory to a new path containing itself.
E$ILLVOL	002DH	The Extended I/O System attempted to physically attach a device that had formerly been only logically attached. In the process, it found that the volume does not contain named files. This prevented the Extended I/O System from completing physical attachment because the named file driver was requested during logical attachment.
E$INVALID$FNODE	003DH	The fnode for the specified file is invalid. The file cannot be accessed; you should delete it.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.

EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
EIOMEM	0042H	The Basic I/O System job does not currently have a block of memory large enough to allow this system call to run to completion.
E$LIMIT	0004H	At least one of the following is true: • The user object or the calling task's job is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job. • The calling task's object limit has been reached. • Processing this call would deplete the remote server's resources. For a list of remote server resources, refer to the *iRMX® Networking Software User's Guide*.
ELOGNAME$-NEXIST	0045H	At least one of the specified paths contains an explicit logical name, but the call was unable to find this name in the object directories of the calling task's local job, the global job, or the root job.
ELOGNAME$-SYNTAX	0040H	At least one of the specified paths contain one or more of the following logical name syntax errors: • A path starts with a colon (:), indicating that it contains a logical name. But the call was unable to find a second colon to delimit the logical name. • A path contains a logical name that is either longer than 12 characters (including colons), has no characters, or contains invalid characters.
E$MEDIA	0044H	The device containing the specified file is not on-line.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
E$NAME$NEXIST	0049H	The user object does not represent a verified user or the user object is not properly defined in the remote server's User Definition File (UDF).

E$NOPREFIX	8022H	At least one of the specified paths contains no explicit prefix (no logical name), and the default prefix for the calling task's job is either undefined, or it is not a valid device connection or file connection.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTFILE$CONN	0032H	In the path pointed to by path$ptr, the subpath portion is null and the prefix portion is not a file connection.
ENOTLOG$NAME	8040H	At least one of the specified paths contains a logical name that refers to an object that is neither a device connection nor a file connection.
ENOTSAME$DEV	003AH	The two paths refer to different devices.
E$NOUSER	8021H	The calling task's job does not have a default user object, or the object cataloged in R?IOUSER is not a user object.
E$PASSWORD$-MISMATCH	004BH	The password of the user object does not match the password of the corresponding user defined on the remote server.
E$PATHNAME$-SYNTAX	003EH	One or both of the specified path names contain invalid characters.
E$PARAM	8004H	The specified task$priority for an IO job is unequal to 0 and is greater than the max$priority of the IO job.
E$SPACE	0029H	At least one of the following is true: • The volume is full. • No more files can be created on the remote server's volume. The Remote File Driver cannot distinguish between an E$FNODE$LIMIT and an E$SPACE condition code.
EUDFIO	02D0H	An error occurred while accessing the remote server's User Definition File (UDF). The server's UDF must have world read permission.
E$SUPPORT	0023H	The task attempted to rename a physical or stream file.

Using the S$SEEK system call, your tasks can move the file pointer for any open physical- or named-file connection. This system call cannot be used with stream files.

```
CALL RQ$S$SEEK(connection, mode, move$count, except$ptr);
```

Input Parameters

connection — A TOKEN for an open connection whose file pointer you wish to move.

mode — A BYTE containing a value that controls the nature of the movement of the file pointer. Any of the following values are valid:

Mode	Meaning
1	Move the pointer backward by the number of bytes specified in move$count. If the move count is large enough to position the pointer past the beginning of the file, the pointer moves to the first byte (position zero).
2	Set the pointer to the position specified by the move count. Position zero is the first position in the file. Moving the pointer beyond the end of the file is valid for named files only.
3	Move the file pointer forward by the specified amount. Moving the pointer beyond the end-of-file is valid for named files.
4	First move the pointer to the end of the file and then move it backward by the specified amount. If the value specified by move$count would position the pointer beyond the front of the file, the pointer moves to the first byte in the file (position zero).

move$count — A DWORD integer that tells the Extended I/O System how far, in bytes, to move the pointer.

Output Parameter

except$ptr — A POINTER to the WORD where the Extended I/O System returns the condition code.

Description

When performing random I/O, your tasks must use this system call to position the file pointer before using the S$READ$MOVE, S$TRUNCATE$FILE, and S$WRITE$MOVE system calls. The location of the file pointer tells the Extended I/O System where in the file to begin reading, truncating, or writing information.

If your tasks are performing sequential I/O on a file, they do not need to use this system call.

Access Control

Two requirements relate to access control. First, the connection must be open for reading only, writing only, or both reading and writing. If this is not the case, your task can use the S$OPEN system call to open the file.

The second access requirement is that the connection must have been created by a task within the calling task's job. If this is not the case, use the existing connection as a prefix, and have the calling task obtain a new connection by invoking the S$ATTACH$FILE system call. This newly created connection satisfies the second requirement.

Reading and Writing Beyond the End of File

It is legitimate to position the file pointer beyond the end-of-file for a named file. If your task does this and then invokes the S$READ$MOVE system call, the Extended I/O System behaves as though the reading operation began at the end-of-file.

Also, it is possible to invoke the S$WRITE$MOVE system call with the file pointer beyond the end of the file. If your task does this, the Extended I/O System attempts to expand the file. If the Extended I/O System does expand your file in this manner, the file contains random information between the old end-of-file and the position of the file pointer when the S$WRITE$MOVE call was invoked.

Condition Codes

E$OK	0000H	No exceptional conditions.
EBADBUFF	8023H	**This condition code is not supported in the iRMX I Operating System.** One of the following is true: • The specified memory buffer is not writeable. • The specified memory buffer crosses a segment boundary.

E$CONN$NOT$OPEN	0034H	At least one of the following is true: • The connection is not open. • The connection was opened by an A$OPEN rather than an S$OPEN.
E$EXIST	0006H	The connection parameter is not a token for an existing object.
E$FLUSHING	002CH	The specified device is being detached.
E$IDDR	002AH	This request is invalid for the device driver. For example, it is not valid to use this call with a line printer.
E$IFDR	002FH	The call attempted to seek in a stream file. The S$SEEK system call can be used only with named and physical files.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOMODE	0056H	A tape drive attempted a read (write) operation before the previous write (read) completed.
EIONO$DATA	0055H	A tape drive attempted to read the next record, but it found no data.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
E$LIMIT	0004H	At least one of the following is true: • Either the calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a file connection.

E$PARAM	8004H	At least one of the following is true: • The value of the mode parameter is not 1, 2, 3, or 4. • The calling task was attempting to seek past the end of a physical file.
E$SPACE	0029H	This seek operation forced the Extended I/O System to attempt to empty the connection's buffer(s) by writing their contents to the volume. However, the volume is full.
E$SUPPORT	0023H	The connection parameter refers to a connection that was created by a task outside of the calling task's job.

The S$SPECIAL system call allows your tasks to perform functions that pertain to a specific device.

```
CALL RQ$S$SPECIAL(connection, function, data$ptr, iors$ptr,
                  except$ptr);
```

Input Parameters

connection — A TOKEN for a connection to the file for which the special function is to be performed.

To access a remote server, this parameter must be a connection to the server's virtual root directory.

function — A WORD that specifies the special function being requested. Each function is described in detail under the "Description" heading, but the following table summarizes the values to be assigned to this parameter.

Only function value 2 (Notify) is supported for remote servers.

Function Value	Type of file for connection	Effect of Function
0	Physical	Format disk track
0	Stream	Query
1	Stream	Satisfy
2	Physical or Named	Notify
3	Physical	Get disk special data
4	Physical	Get terminal data
5	Physical	Set terminal data
6	Physical	Set signal character
7	Physical	Rewind tape
8	Physical	Read tape file mark
9	Physical	Write tape file mark
10	Physical	Retention tape
11-15		Reserved for other Intel products
16	Physical	Get terminal status
17	Physical	Cancel terminal I/O
18	Physical	Resume terminal I/O
19-32767		Reserved for other Intel products
32768-65555		Reserved for user devices

data$ptr — A POINTER to a parameter block that your task uses to supply the Extended I/O System with information, or to receive information from the Extended I/O System. The contents and form of the parameter block depend upon the function being requested, so the form of the parameter block is described later, under the "Description" heading. If the function requires no parameter block, set data$ptr to NIL.

Output Parameters

iors$ptr — A POINTER to a structure of the form described below. The Extended I/O System uses this structure to return information that might be of use to the calling task. If you set this POINTER to NIL, the Extended I/O System does not return the information. Be aware that this is relatively obscure information that most applications do not need.

```
DECLARE iors$data STRUCTURE(
                    actual          WORD,
                    actual$fill     WORD,
                    device          WORD,
                    unit            BYTE,
                    funct           BYTE,
                    subfunct        WORD,
                    device$loc      DWORD,
                    buf$ptr         POINTER,
                    count           WORD,
                    count$fill      WORD,
                    aux$ptr         POINTER)
```

where:

actual	Number of bytes that were actually transferred during the special function, if any.
actual$fill	Reserved for Intel's use.
device	Device number identifying the device. For an explanation of device numbers, refer to the configuration reference manual included with the operating system.
unit	Number of the unit that contains the file on which the special function is being performed.
funct	Code recognized by the driver, usually meaning that this is a special operation.
subfunct	Function code you code into the call.
device$loc	Location on the device where the operation was performed.
buf$ptr	POINTER to a buffer used for this operation, if any buffer is used.
count	Number of bytes transferred, if any were transferred.
count$fill	Reserved for future use.
aux$ptr	Same as data$ptr in the call to S$SPECIAL.

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

This system call allows your tasks to communicate with devices, device drivers, and the stream file driver to perform operations that are less device-independent than other Extended I/O System operations.

S$SPECIAL allows your task to perform several special functions. The Extended I/O System decides which function to perform by examining the function parameter and the kind of connection provided in the connection parameter. The following sections explain each function in detail.

Special Considerations for iRMX®-NET

iRMX-NET only supports function value 2 (Notify) for remote servers. When a task invokes S$SPECIAL with a connection to a remote server and function equal to 2, the calling task is notified of a communication failure immediately after an unsuccessful attempt to access the remote server, or when the device connection to the remote server is physically detached. Communication failures can result from resetting the server, faults in the consumer or server, or line transmission errors.

To restore the availability of a remote server, perform the following steps:

1. Fix the communication problem.
2. Call A$PHYSICAL$DETACH$DEVICE to detach the server's device connection.
3. Call A$PHYSICAL$ATTACH$DEVICE to reattach the server.

Formatting a Track (Function Code 0)

To use the S$SPECIAL system call to format a track on a disk, the calling task must supply the following information:

connection	A TOKEN for a connection to a physical file. This connection must be open for reading, writing or both.
function	Must be set to zero.

data$ptr — Must point to a STRUCTURE of the following form:

```
DECLARE track$formatter STRUCTURE(
                    track$number        WORD,
                    interleave          WORD,
                    track$offset        WORD,
                    fill$char           BYTE)
```

where:

track$number — Number of the track to be formatted. Acceptable values are 0 to one less than the number of tracks on the volume. Other values cause an E$SPACE exception code. When formatting a tape or a RAM-disk, you must place a zero value in this field.

interleave — The number of physical sectors between consecutive logical sectors. (This field does not apply to tapes or to RAM-disks.) If the interleave factor is zero or one, no physical sectors are skipped. If the specified interleave value is greater than the number of physical sectors on a track, the operating system divides that interleave value by the number of physical sectors and uses the remainder as the interleave factor. A remainder of zero has the same effect as an interleave of zero.

track$offset — Number of physical sectors to skip between the index mark and the first logical sector. (This field does not apply to tapes or to RAM-disks.)

fill$char — The character with which the sector will be written; some drivers ignore this field and fill the sectors with a character the driver establishes.

Also see the description of Function Code 3, Getting Special Disk Data.

Obtaining Information About Stream File Operations (Function Code 0)

Occasionally, a task using a stream file must find out what is being requested by another task using the same stream file. For example, the task reading a stream file might need to know how many bytes are being sent by a task writing to the same file. Tasks can obtain this kind of information by calling S$SPECIAL with the following information:

connection	A TOKEN for a connection to a stream file.
function	Set to 0.
data$ptr	Set to NIL.

If a task is reading from or writing to a stream file, the Extended I/O System returns information in the structure to which iors$ptr points. The following four fields contain valid information:

actual	The number of bytes already transferred.
count	The number of bytes remaining to be transferred.
buf$ptr	A POINTER to the memory location to be used for the next byte to be transferred.
funct	A value that indicates the purpose of the queued request. The value is zero for read requests and one for write requests.

If no task is reading from or writing to the stream file, the Extended I/O System queues the S$SPECIAL request. The request remains queued until a task issues a read or write request. If, before a read or write request is issued, another S$SPECIAL request arrives, the Extended I/O System cancels both S$SPECIAL requests and returns E$STREAM$SPECIAL exception codes to the tasks that issued the S$SPECIAL calls.

Satisfying Stream File Transactions (Function Code 1)

Stream files provide two tasks with the ability to communicate. When one task tries to read or write to a stream file, the task does not run again until the complementary task issues a matching request.

For example, suppose that Task A wants to read 512 bytes, but Task B writes only 256 bytes. Task A stops running until Task B issues one or more requests which supply at least 256 more bytes.

The S$SPECIAL system call enables tasks to force a stream file transaction to complete, even if the number of bytes written does not match the number of bytes read.

To force this completion, a task must invoke the S$SPECIAL system call with the parameters set as follows:

connection	A TOKEN for a connection to the stream file. This connection must be open for the operation that has not satisfied the matching requirement. For example, if the reading task wants to force the Extended I/O System to consider the transaction completed, the connection must be open for reading.
function	Set to 1.
data$ptr	Set to NIL.

After requesting this satisfy function, the only information that your task can obtain is the condition code returned by the Extended I/O System. If the task invoking the S$SPECIAL system call has already completed the transaction, the Extended I/O System returns an E$STREAM$SPECIAL condition code.

Requesting Notification that a Volume is Unavailable (Function Code 2)

This function applies to named and physical files only. When a person opens a door to a flexible disk drive or presses the STOP button on other mass storage drives, the volume mounted on that drive becomes unavailable. A task can request notification of such an event by calling S$SPECIAL. For flexible disk drives attached to an iSBC 208 or iSBC 218A controller, and for some 5-1/4" flexible disk drives, notification occurs when the Basic I/O System first tries to perform an operation on the unavailable volume. For most other drives, notification occurs immediately. The reason for this difference is that controller/drive combinations that include the iSBC 208 or iSBC 218A controller, or that include some 5-1/4" drives, cannot generate an interrupt when the drive ceases to be ready. In contrast, most other controller/drive combinations do.

On those drives where no notification occurs until the Basic I/O System attempts to access the drive, a dangerous situation occurs whenever you change a volume without first detaching the device. If you do not first detach the device and then reattach it, the Basic I/O System accesses the device using the directory information from the old volume. Unless the new volume is write-protected, this process corrupts the entire volume, rendering it useless. The correct sequence of events when changing volumes on one of these devices is as follows:

1. Detach the unit (via A$PHYSICAL$DETACH$DEVICE).
2. Remove the old volume.
3. Install the new volume.
4. Reattach the unit (via A$PHYSICAL$ATTACH$DEVICE).

For devices that can perform notification, a task requests notification by calling S$SPECIAL with a token for a device connection, with function set to 2, and with data$ptr pointing to a structure of the following form:

```
DECLARE notify      STRUCTURE(
                        mailbox     TOKEN,
                        object      TOKEN);
```

where:

mailbox	A TOKEN for a mailbox.
object	A TOKEN for an object. When the Basic I/O System detects that the implied volume is unavailable, the object is sent to the mailbox.

After a task has made a request for notification, the Basic I/O System remembers the object and mailbox tokens until either the volume is detected as being unavailable or until the device is detached by the A$PHYSICAL$DETACH$DEVICE system call. When the volume becomes unavailable, the object is sent to the mailbox. Note that this implies that some task should be dedicated to waiting at the mailbox.

If the volume is detected as being unavailable, the Basic I/O System will not execute I/O requests to the device on which the volume was mounted. Such requests are returned with the status field of the I/O request/result segment set to E$IO and the unit$status field set to IO$OPRINT (value = 3). The latter code means that operator intervention is required.

If any task issues a subsequent notification request for the same device connection, the Basic I/O System replaces the old mailbox and object values with the new ones specified. It does not return an exception code.

To restore the availability of a volume, perform the following steps:

1. Close the door of the diskette drive or restart the hard disk drive.
2. Call A$PHYSICAL$DETACH$DEVICE. It may be necessary to do a "hard" detach of the device.
3. Call A$PHYSICAL$ATTACH$DEVICE and reattach the device.
4. Create a new file connection.

To cancel a request for notification, make a dummy request using the same connection with a SELECTOR$OF(NIL) value in the mailbox parameter.

Getting Disk Special Data (Function Code 3)

You can write your own program to format a disk, rather than using the FORMAT command (part of the iRMX Human Interface). If you do so, you must place some special device data into the last bytes of the label on the iRMX named volume. Currently, this field in the label is eight (8) bytes long, although Intel reserves the right to add to its length later. (The structure of an iRMX named file volume is described in the *iRMX® Disk Verification Utility Reference Manual.*)

You can obtain the data in this field by issuing S$SPECIAL with a function code of three. You can then save the data and write it into the label field when you format the disk.

To use the S$SPECIAL system call to obtain the special data for the label, the calling task must supply the following information:

connection	A TOKEN for a connection to a physical file. This connection must be open for reading, for writing, or for both reading and writing.
function	Set to 3.
data$ptr	A POINTER to a STRUCTURE of the following form:

```
DECLARE disk$label$data     STRUCTURE(
        label$data(8)               WORD);
```

Getting Terminal Characteristics (Function Code 4)
Setting Terminal Characteristics (Function Code 5)

These two functions are complements of each other. They use the same type of data structure with identical meanings for each field in the structure. A function code of four returns the current characteristics of a particular terminal; a function code of five allows you to set the characteristics of a terminal.

Intel recommends that before setting the terminal characteristics, you first invoke S$SPECIAL with function code 4 to get the current characteristics. Then, modify the returned structure to reflect your desired changes. Finally, invoke S$SPECIAL with function code 5 to set the characteristics, using your modified structure as input.

In this section, certain terms unique to terminal devices (for example, line editing, Operating System Command (OSC) sequences, translation) are described only briefly. If you are unfamiliar with these terms refer to the *iRMX® Basic I/O System Calls Reference Manual* and the *iRMX® Device Driver User's Guide.*

To use the S$SPECIAL system call to get or to set terminal characteristics, the calling task must supply the following information:

connection — A TOKEN for a connection to a terminal.

function — Set to 4 (get characteristics) or 5 (set characteristics).

data$ptr — A POINTER to a STRUCTURE of the following form:

```
DECLARE terminal$attributes STRUCTURE(
                num$words           WORD,
                num$used            WORD,
                connection$flags    WORD,
                terminal$flags      WORD,
                in$baud$rate        WORD,
                out$baud$rate       WORD,
                scroll$lines        WORD,
                x$y$size            WORD,
                x$y$offset          WORD,
                special$modes       WORD,
                high$water$mark     WORD,
                low$water$mark      WORD,
                fc$on$char          WORD,
                fc$off$char         WORD,
                link$parameter      WORD,
                spc$hi$water$mark   WORD,
                special$char(4)     BYTE);
```

where:

num$words — The number of words, not including num$words and num$used, that are reserved for the remainder of the terminal$attributes data structure. To access all of the information, set this field to at least 16. Intel reserves the right to expand the length of this structure in later releases.

num$used — The number of fields, following the num$used field, that are actually being used for getting or setting terminal characteristics.

In getting and setting terminal information, the amount of data returned or sent is governed by the num$used field. For example, if function is 4 and num$used is 2, then an S$SPECIAL call returns data in the connection$flags and terminal$flags fields, but not in the remainder of the fields.

However, when setting terminal attributes, specifying a zero value for any of the next five

fields (connection$flags through scroll$lines) causes the I/O System to skip over the zeroed field, leaving it at its previous setting. For example, if num$used is 2, while connection$flags is 0 and terminal$flags is not 0, then S$SPECIAL uses the contents of the terminal$flags field to set terminal attributes, but it ignores the contents of connection$flags field. In this way, you can set some parameters without affecting others.

For the functions represented by the remaining fields in this structure, invoking S$SPECIAL is not the only way to set the functions. You can also set them with OSC sequences. The description of each field mentions, in parentheses, the OSC characters you can use. (OSC sequences are described in the *iRMX® Device Drivers User's Guide*.) You can also use the OSC Query sequence when debugging, to ensure that your tasks invoked S$SPECIAL correctly.

connection$flags

This word applies only to this connection to the terminal. (All other parameters apply to the terminal itself and therefore to all connections to the terminal.) If you attempt to set this field to zero, the I/O System ignores your entry and leaves the field set to its previous value.

Changes you make with connection$flags don't take effect until a read is processed using the connection. Therefore, to ensure the changes take effect, you should read from the connection immediately after using connection$flags to change the connection's attributes. (If you don't expect input at the terminal, set the connection to flush mode, then read 255 characters from the connection. The read will return immediately with whatever characters were available.)

The flags in this word are encoded as follows. (Bit 0 is the low-order bit.)

Bits	Value and Meaning
0-1	Line editing control (corresponds to OSC characters C:T). Line editing refers to how the TSC (Terminal Support Code) handles control characters such as those that delete characters entered at a terminal, scroll terminal output, and others. Refer to the *iRMX® Device Drivers User's Guide* for more information.

NOTE

Line editing is supported on input only (that is, the stream of data entered at, but not sent to, a terminal).

Bits	Value and Meaning
	0 = Invalid Entry.
	1 = Transparent mode (no line editing). Input is transmitted to the requesting task exactly as entered at the terminal*. Before being transmitted, data accumulates in a buffer until the requested number of characters has been entered.
	2 = Normal mode (line editing). Edited data accumulates in a buffer until a line terminator is entered.
	3 = Flush mode (no line editing). Input is transmitted to the requesting task exactly as entered at the terminal*. Before being transmitted, data accumulates in a buffer until an input request is received. At that time, the contents of the buffer (or the number of characters requested, if the buffer contains more than that number) is transmitted to the requesting task. If any characters remain in the buffer, they are saved for the next input request.
2	Echo control (corresponds to OSC characters C:E).
	0 = Echo. Characters entered at the terminal are "echoed" to the terminal's display screen.
	1 = Do not echo.

* Except (1) signal characters (e.g., the Human Interface CONTROL-C) set by specifying "set signal" in the function parameter of A$SPECIAL or S$SPECIAL, and (2) any enabled output control characters or OSC sequences.

3	Input parity control (corresponds to OSC characters C:R). Characters entered into the terminal have their parity bits (bit 7) set to 0 or not set by the Terminal Support Code, according to the value of the input parity control bit. 0 = Set parity bit to 0. 1 = Do not alter parity bit.
4	Output parity control (corresponds to OSC characters C:W). Characters being output to the terminal have their parity bits (bit 7) set to 0 or not set by the Terminal Support Code, according to the value of the output parity control bit. 0 = Set parity bit to 0. 1 = Do not alter parity bit.
5	Output control character control (corresponds to OSC characters C:O). This bit specifies whether output control characters are effective when entered at the terminal. The value of this bit applies only to output through this connection. Control characters are described in the *iRMX® Device Drivers User's Guide*. Note that the output control characters are supported only on input from a terminal, not as output to a terminal. 0 = Accept output control characters in the input stream. 1 = Ignore output control characters in the input stream.

6-7	OSC control sequence enable/disable (corresponds to OSC characters C:C). These bits specify whether OSC control sequences should be acted upon when they appear in the input stream and, separately, when they appear in the output stream. These bits apply only to input or output through this connection. OSC control sequences are described in *iRMX® Device Drivers User's Guide*. 0 = Act upon OSC sequences that appear in either the input or output stream. 1 = Act upon OSC sequences in the input stream only. 2 = Act upon OSC sequences in the output stream only. 3 = Do not act upon any OSC sequences.
8	Specifies whether characters in the raw input buffer are moved to the type-ahead buffer by the interrupt task or the service task. The raw input and type-ahead buffers are discussed in the *iRMX® Device Drivers User's Guide*. 0 = Characters are moved from the raw input buffer to the type-ahead buffer by the interrupt task. 1 = Characters are moved from the raw input buffer to the type-ahead buffer by the service task.
9	Specifies whether the type-ahead buffer is used to process characters in the raw input buffer. 0 = Characters are moved from the raw input buffer to the type-ahead buffer. 1 = Characters are moved directly from the raw input buffer to the application task's buffer, thus bypassing the type-ahead and line-edit buffers. This disables all Terminal Support Code features.
10-15	Reserved bits. For future compatibility, set to 0.

terminal$flags

This word applies to the terminal and therefore to all connections to the terminal. If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The flags in this word are encoded as follows. (Bit 0 is the low-order bit.)

Bits	Value and Meaning
0	Reserved bit. Set to 1.
1	Line protocol indicator (corresponds to OSC characters T:L). Full-duplex terminals support simultaneous and independent input and output. Half-duplex terminals support independent input and output, but not simultaneously. 0 = Full duplex. 1 = Half duplex.
2	Output medium (corresponds to OSC characters T:H). 0 = Video display terminal (VDT). 1 = Printed (Hard copy).
3	Modem indicator (corresponds to OSC characters T:M). 0 = Not used with a modem. 1 = Used with a modem.

4-5

Input parity control bits (corresponding to OSC characters T:R) determines how the terminal driver handles input parity. The parity bit (bit 7) of each input byte can be used in a variety of ways. A byte has even parity if the sum of its bits is an even number. Otherwise, the byte has odd parity.

0 = Terminal driver always sets parity bit to 0.

1 = Terminal driver never alters the parity bit.

2 = Even parity is expected on input. The terminal driver uses the parity bit to indicate the presence (1) or absence (0) of an error on input. That is, the driver sets the parity bit to 0 unless the received byte has odd parity or there is some other error, such as (a) the received stop bit has a value of 0 (framing error) or (b) the previous character received has not yet been fully processed (overrun error).

3 = Odd parity is expected on input. The terminal driver uses the parity bit to indicate the presence (1) or absence (0) of an error on input. That is, the driver sets the parity bit to 0 unless the received byte has even parity or there is some other error, such as (a) the received stop bit has a value of 0 (framing error) or (b) the previous character received has not yet been fully processed (overrun error).

6-8 Output parity control bits (corresponding to OSC characters T:W). Determines how the terminal driver handles output parity. The parity bit (bit 7) of each output byte can be used in a variety of ways. A byte has even parity if the sum of its bits is an even number. Otherwise, the byte has odd parity.

- 0 = Terminal driver always sets parity bit to 0.
- 1 = Terminal driver always sets parity bit to 1.
- 2 = Terminal driver sets parity bit to give the byte even parity.
- 3 = Terminal driver sets parity bit to give the byte odd parity.
- 4 = Terminal driver does not alter the parity bit.
- 5-7 Invalid values.

9 Translation control (corresponds to OSC characters T:T). Translation refers to the ability to define certain control characters so that whenever these characters are entered at or written to a terminal, certain actions, usually cursor movements, take place automatically. Translation is described in the *iRMX® Device Drivers User's Guide*.

- 0 = Do not enable translation.
- 1 = Enable translation.

10 Terminal axes sequence control (corresponds to OSC characters T:F). This specifies the order in which Cartesian-like coordinates of elements on a terminal's screen are to be listed or entered.

- 0 = List or enter the horizontal coordinate first.
- 1 = List or enter the vertical coordinate first.

11	Horizontal axis orientation control (corresponds to OSC characters T:F). This specifies whether the coordinates on the terminal's horizontal axis increase or decrease as you move from left to right across the screen. 0 = Coordinates increase from left to right. 1 = Coordinates decrease from left to right.
12	Vertical axis orientation control (corresponds to OSC characters T:F). This specifies whether the coordinates on the terminal's vertical axis increase or decrease as you move from top to bottom across the screen. 0 = Coordinates increase from top to bottom. 1 = Coordinates decrease from top to bottom.
13-15	Reserved bits. For future compatibility, set to 0.

NOTE

If bits 4-5 contain 2 or 3, and bits 6-8 also contain 2 or 3, then they must both contain the same value. That is, they must both reflect the same parity convention (even or odd).

in$baud$rate	The input baud rate indicator (corresponds to OSC characters T:I). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows: 0 = Leave field set to the previous value. 1 = Use the input baud rate for output. Other = Actual output baud rate, such as 9600.
out$baud$rate	The output baud rate indicator (corresponds to OSC characters T:O). If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value. The word is encoded as follows: 0 = Leave field set to the previous value. 1 = Use the input baud rate for output. Other = Actual output baud rate, such as 9600.

	Most applications require the input and output baud rates to be equal. In such cases, use in$baud$rate to set the baud rate and specify a one for out$baud$rate.
scroll$lines	An operator at a terminal can enter a control character (default is CONTROL-W) when he/she is ready for data to appear on the terminal's display screen. The scroll$lines value (corresponding to OSC characters T:S) specifies the maximum number of lines that are to be sent to the terminal each time the operator enters the control character. If you attempt to set this field to zero, the Basic I/O System ignores your entry and leaves the field set to its previous value.
x$y$size	The low-order byte of this word specifies the number of character positions on each line of the terminal's screen (and corresponds to OSC characters T:X). The high-order byte specifies the number of lines on the terminal's screen (and corresponds to OSC characters T:Y).
x$y$offset	The low-order byte of this word specifies the value that starts the numbering sequence of both the X and Y axes (and corresponds to OSC characters T:U). The high-order byte specifies the value to which the numbering of the axes must "fall back" after reaching 127 (and corresponds to OSC characters T:V).
special$modes	This and the following fields apply only to buffered devices (such as the iSBC 544A and the iSBC 188/48 boards). These devices maintain their own input and output buffers separately from the ones managed by the Basic I/O System's Terminal Support Code. If you aren't sure whether you can set these fields, invoke S$SPECIAL with function code 4 to get the terminal attributes. If bit 15 of the special$modes field is set, your board is a buffered device and you can set the bits in special$modes and the following fields. (If your board is not a buffered device, setting any of the following fields will cause the Terminal Support Code to return an E$PARAM Condition Code.)

Bits	Value and Meaning
0	Flow control mode specifies whether the communications board sends flow control characters (selected by the fconchar and fcoffchar fields, but usually XON and XOFF) to turn input on and off (corresponds to the OSC characters T:G). The low-order bit (bit 0) controls this option, as follows: 0 = Disable flow control. 1 = Enable flow control. When flow control is enabled, the communication board can control the amount of data sent to it to prevent buffer overflow. This is especially important when communicating with another computer.
1	With the Special Character Mode (corresponds to OSC characters T:D) you can define up to four special characters. These special characters are different from the signal characters provided by the Terminal Support Code, though they may be signal characters. If your driver supports special characters, it processes these characters differently when the Special Character Mode is on. 0 = Disable Special Character Mode. 1 = Enable Special Character Mode.
2-14	Reserved bits. Set to 0.
15	Buffered Device Control. This bit is set by the Terminal Support Code to show if a device is buffered. If invoking the S$SPECIAL system call to get terminal attributes shows that this bit is set, then the special$modes bits and the data fields following are valid. If the Buffered Device Control bit is not set and you attempt to alter these data fields, an E$PARAM error is returned 0 = Not a buffered device. 1 = Buffered device.

The remaining fields in the structure apply only to buffered devices.

high$water$mark — When the communication board's buffer fills to contain the number of bytes represented by this field, the board's firmware sends the flow control XOFF character to stop input. (This field corresponds to the OSC characters T:J.)

The high-water mark of the iSBC 544A board is not configurable; therefore, setting this field has no effect on that board.

low$water$mark — When the number of bytes in the communication board's input buffer drops to the number represented by this field, the board's firmware sends the flow control "on" character to start input. (This field corresponds to the OSC characters T:K.)

The low-water mark of the iSBC 544A board is not configurable; therefore, setting this field has no effect on that board.

fconchar — An ASCII character that the communication board sends to the connecting device when the number of bytes in its input buffer drops to the low-water mark. Normally this character tells the connecting device to resume sending data. (This field corresponds to the OSC characters T:P.)

The fconchar for the iSBC 544A board is set to the XON (CONTROL-Q) character and is not configurable; therefore, setting this field has no effect on that board.

fcoffchar — An ASCII character that the communication board sends to the connecting device when the number of characters in its input buffer rises to the high-water mark. Normally this character tells the connecting device to stop sending data. (This field corresponds to the OSC characters T:Q.)

The fcoffchar for the iSBC 544A board is set to the XOFF (CONTROL-S) character and is not configurable; therefore, setting this field has no effect on that board.

link$parameter — (Corresponds to the OSC characters T:N) This word specifies the characteristics of the physical link between the terminal and a device. Not all device drivers support link$parameter. This field is supported by those boards supported by the Terminal Communications Controller driver, including the iSBC 188/48, iSBC 188/56, iSBC 546, iSBC 547, iSBC 548, and iSBC 548 controllers.

The meaning of the bits in this field are as follows:

Bits	Value and Meaning
0-1	Parity 0 = No parity 1 = Invalid Value 2 = Even parity 3 = Odd parity
2-3	Character length 0 = 6 bits/character. 1 = 7 bits/character. 2 = 8 bits/character. 3 = Invalid Value
4-5	Number of stop bits. 0 = 1 stop bit. 1 = 1 1/2 stop bits. 2 = 2 stop bits.
6-14	Reserved
15	Check if this word is to be used 0 = not used 1 = used

If parity is enabled, an additional bit position beyond those specified in the Character Length control is added to the transmitted data and expected in received data. The received parity bit is transferred to the CPU as part of the data unless 8 bits/character is selected. If a parity error is detected on input, the character is discarded.

In the 6 and 7 bits/character modes unused bit positions in transmit data are ignored. Unused bits in receive data are set to 1. If a framing error is detected on input, the character is returned as an 8-bit null (00H).

Bit 15 is checked to see if this word is to be used. If set to 1, the driver passes the low-order byte to the controller, which sets the parity, character length, and stop bits. If set to 0, this word is skipped and the terminal$flags field is used.

spchiwater$mark — This word specifies the high-water mark used by the special character mode (bit 1 of special$modes) and is ignored if the special character mode is off. If your device driver supports the special character mode, the driver processes special characters differently when the number of characters in the input buffer reaches the high-water mark. You can define up to four special signal characters (corresponds to the OSC characters T:A).

special$char(4) — This array holds the characters you define as special characters (and corresponds to the OSC characters T:Z). If you define less than four special characters, then you must fill the remaining slots in the array with duplicates of the last character you define.

Designating Characters for Signaling from a Terminal Keyboard (Function Code 6)

You can use the S$SPECIAL system call to associate a keyboard character with a semaphore, so that whenever the character is entered into the terminal, the Basic I/O System automatically sends a unit to the semaphore. Up to 12 character-semaphore pairs can be so associated simultaneously; each character being associated with a different semaphore, if desired. Character-semaphore pairs are called Signal Characters.

To set up a signal character, call S$SPECIAL with a device connection, with function equal to 6, and with data$ptr pointing to a structure of the following form:.i.structure:signal$pair;

```
DECLARE signal$pair STRUCTURE(
                          semaphore       TOKEN,
                          character       BYTE);
```

where:

semaphore — A TOKEN for the semaphore that is to be associated with the character.

character — If the character value is in the range 0 to 1FH, or is 7FH, the terminal support code sends a unit to the associated semaphore when it receives the ASCII equivalent of this value.

If you add 20H to the character values in the 0 to 1FH range (making this range 20H to 3FH), or if the value is 40H, then the type ahead buffer (and the input buffer if this is a buffered device) is cleared and a unit is sent to the associated semaphore.

To delete a signal character, call S$SPECIAL with the semaphore field set to 0 and character set to the signal character to be deleted.

Tape Drive Functions (Function Codes 7, 8, 9, and 10)

The S$SPECIAL system call performs four different functions that apply to tape drives only. These functions include rewinding a tape, searching for file marks, writing file marks, and retentioning a tape.

To rewind a tape, call S$SPECIAL with the following information:

connection — A TOKEN for a connection to a physical file.

function — Set to 7.

data$ptr — Set to NIL.

This function terminates tape read and write operations and rewinds a tape to its load point. If the tape drive is performing a write operation when you invoke this call, the tape drive writes a file mark before it rewinds the tape.

To search for a file mark, call S$SPECIAL with the following information:

connection	A TOKEN for a connection to a physical file.
function	Set to 8.

data$ptr Set to NIL.This function terminates tape read operations and moves the tape to the next file mark. Any outstanding requests are completed before this call takes effect.

To write a file mark, call S$SPECIAL with the following information:

connection	A TOKEN for a connection to a physical file.
function	Set to 9.
data$ptr	Set to NIL.

This function terminates tape write operations and writes a file mark at the current position on the tape.

To retention a tape, call S$SPECIAL with the following information:

connection	A TOKEN for a connection to a physical file.
function	Set to 10.
data$ptr	Set to NIL.

This function fast-forwards the tape to the end and then rewinds it to the load point.

Getting Terminal Status (Function Code 16)

This function applies only to physical files. You can get the status of a terminal that is being driven by a terminal device driver by issuing a call to S$SPECIAL.

In this section, certain terms unique to terminal devices (for example, line-editing, OSC sequences, translation) are described only briefly. If you are unfamiliar with these terms, refer to the *iRMX® Device Drivers User's Guide*.

To get a terminal's status, call S$SPECIAL with a connection for the terminal, with function equal to 16, and with data$ptr pointing to a structure of the form:

```
DECLARE terminal$status STRUCTURE(
                terminal$flags          WORD,
                input$conn$flags        WORD,
                input$state             WORD,
                input$conn              TOKEN,
                input$count             WORD,
                input$actual            WORD,
                raw$buf$count           WORD,
                typeahead$count         BYTE,
                num$input$requests      BYTE,
                output$conn$flags       WORD,
                output$state            WORD,
                output$conn             TOKEN,
                output$count            WORD,
                output$actual           WORD,
                out$buf$count           WORD,
                num$output$requests     BYTE);
```

where:

terminal$flags — The current attributes associated with the terminal. For the meaning of the bits in this word, see the terminal$flags parameter in the description of function codes 4 and 5 of the S$SPECIAL system call.

input$conn$flags — The current attributes associated with the terminal's active input connection. For the meaning of the bits in this word, see the connection$flags parameter in the description of function codes 4 and 5 of the S$SPECIAL system call.

input$state — The internal state of this terminal's input connection. The bits in this WORD are encoded as follows. (Bit 0 is the low-order bit.)

Bits	Value and Meaning
0	Indicates whether an input request has been set up. 0 = An input request has not been set up. 1 = An input request has been set up.
1	Indicates whether the current input request has completed. 0 = The current input request has not completed. 1 = The current input request has completed.
2	Reserved
3	Indicates whether an Operating System Command (OSC) sequence is being processed. 0 = An OSC sequence is not being processed. 1 = An OSC sequence is being processed.
4	Indicates whether a complete line has been processed and is ready for transfer from the line-edit buffer to theapplication task's buffer. Only applies to terminals in line-edit mode. 0 = A complete line has not been processed. 1 = A complete line has been processed.
5	Indicates whether the current character was preceded by a CONTROL-P (quoting character) and is being interpreted as data, rather than as a line-editing character. Output control characters, such as CONTROL-S and CONTROL-Q perform their normal functions even if preceded by CONTROL-P. Only applies to terminals in line-edit mode. 0 = The current character was not preceded by a CONTROL-P. 1 = The current character was preceded by a CONTROL-P.

6	Indicates whether an escape sequence is being processed. 0 = An escape sequence is not being processed. 1 = An escape sequence is being processed.
7	Indicates whether a CONTROL-R is being used to recall the last line. Only applies to terminals in line-edit mode. 0 = The last line is not being recalled. 1 = The last line is being recalled.
8	Indicates whether this terminal is on-line and available for use. Only applies to terminal configured for use with a modem. 0 = The terminal is not available for use. 1 = The terminal is available for use.
9	Indicates whether this terminal is waiting for a ring interrupt as a result of a modem query OSC command. Only applies to terminals configured for use with a modem. 0 = The terminal is not waiting for a ring interrupt. 1 = The terminal is waiting for a ring interrupt.
10	Indicates whether this terminal is waiting for a carrier loss interrupt as a result of a modem query OSC command. Only applies to terminals configured for use with a modem. 0 = The terminal is not waiting for a carrier loss interrupt. 1 = The terminal is waiting for a carrier loss interrupt.

	11	Indicates whether this terminal has a modem query pending as a result of a modem query OSC command. Only applies to terminals configured for use with a modem. 0 = The terminal does not have a modem query pending. 1 = The terminal does have a modem query pending.
	12, 13	Reserved.
	14	Indicates whether the current line has been cancelled. Only applies to terminals in line-edit mode. 0 = The current line has not been cancelled. 1 = The current line has been cancelled.
	15	Indicates whether the type-ahead buffer is full. 0 = The type-ahead buffer is not full. 1 = The type-ahead buffer is full.
input$conn	A TOKEN for the most recently used input connection associated with this terminal.	
input$count	The number of characters requested by the latest input request.	
input$actual	The number of characters that were moved from the raw input or type-ahead buffer to the application task's buffer during the latest request.	
rawbufcount	The number of characters available in the raw input buffer.	
typeahead$count	The number of characters available in the type-ahead buffer.	
num$input$requests	The number of input requests in the input queue for this terminal.	
output$conn$flags	The current attributes associated with the terminal's active output connection. For the meaning of the bits in this word, see the connection$flags parameter in the description of function codes 4 and 5 of the S$SPECIAL system call.	
output$state	The internal state of this terminal's output connection. This parameter can be used to determine if a terminal's output is hindered in some way (for example, because an XOFF was received). To check for hindered output, AND output$state with the value 1E0H. If the result is non-zero, output is hindered. You can resume terminal output by invoking S$SPECIAL with function code 18.	

The bits in this WORD are encoded as follows. (Bit 0 is the low-order bit.)

Bits	Value and Meaning
0-1	0 = Output character processing is occurring normally without an escape character being encountered.
	1 = An ESC character has been encountered in the output stream. This requires special handling because it may be part of an escape or OSC sequence or it may need to be translated.
	2 = The previously encountered escape character is part of an OSC sequence that is being processed.
	3 = The previously encountered escape character is part of an escape sequence that is being translated.
2	Indicates whether an output request has been set up.
	0 = An output request has not been set up.
	1 = An output request has been set up.
3	Indicates whether the terminal controller is transmitting characters from the current output request or is ready to transmit a character from the next output request. Only applies to non-buffered devices.
	0 = The terminal controller is busy transmitting characters from the current request on an interrupt-driven basis.
	1 = The terminal controller is ready to transmit a character once the next output request arrives.
4	Reserved.
5	Indicates whether this terminal's output is being discarded (in discarding mode).
	0 = Not in discarding mode.
	1 = In discarding mode.

	6	Indicates whether this terminal's output is blocked because an XOFF was received or a page scroll has completed (placing output into stopped mode). 0 = Output is not blocked. 1 = Output is blocked.
	7	Indicates whether this terminal's output is in scroll mode. 0 = Not in scroll mode. 1 = In scroll mode.
	8	Indicates whether the terminal's output is blocked because an XOFF was received. 0 = Output is not blocked. 1 = Output is blocked.
	9	Indicates whether the terminal's current output request has been cancelled and is being flushed. 0 = The terminal request has not been cancelled. 1 = The terminal request has been cancelled.
	10-15	Reserved.
output$conn	A TOKEN for the most recently used output connection associated with this terminal.	
output$count	The number of characters requested by the latest output request.	
output$actual	The number of characters moved from the application task's buffer into the output buffer during the latest output request.	
outbufcount	The number of characters still awaiting output from the output buffer of the Terminal Support Code or the buffered device.	
num$out-put$requests	The number of output requests in the output queue for this terminal.	

Cancelling Terminal I/O (Function Code 17)

The S$SPECIAL system call allows a program to cancel all requests associated with a specified connection to a terminal.

To cancel all requests associated with a connection to a terminal, call S$SPECIAL with a connection for the terminal, with function equal to 17, and with data$ptr pointing to a structure of the form:

```
DECLARE cancel$io$struc STRUCTURE( cancel$conn$t   TOKEN);
```

where:

cancel$conn$t — A TOKEN for the connection whose requests are to be cancelled. Setting cancel$conn$t to SELECTOR$OF(NIL) cancels all input requests associated with the connection specified by S$SPECIAL's connection parameter. To determine which connection is active and can be cancelled, invoke S$SPECIAL with function equal to 16 (get terminal status) and check the TOKEN returned in the input$conn parameter.

NOTE

The cancel terminal I/O function cancels all requests that are using the specified connection. Therefore, unless you have a reason to do otherwise, each task using a particular terminal device should have its own connection to the device. Then the requests associated with a private connection can be cancelled without affecting other input requests on the same terminal device.

Resuming Terminal I/O (Function Code 18)

The S$SPECIAL system call allows a program to resume an output request that is blocked because an output control character was entered at the terminal. To resume an output request that is blocked, call S$SPECIAL with any connection for the blocked terminal and with function equal to 18. The data$ptr parameter is ignored.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONN$NOT$OPEN	0034H	At least one of the following is true: • The connection is not open. • The connection was opened by A$OPEN rather than S$OPEN.
E$EXIST	0006H	The connection parameter is not a token for an existing object.
E$FLUSHING	002CH	The specified device is being detached.
E$IDDR	002AH	The requested function is not supported by the device containing the specified file.
E$IFDR	002FH	The Extended I/O System does not support the requested function for the file driver associated with the connection.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOMEM	0042H	The BIOS memory pool on the remote server does not have a block of memory large enough to allow the system call to run to completion.
EIOMODE	0056H	One of the following is true: • A tape drive attempted to perform a read operation before the previous write operation completed. • A tape drive attempted to perform a write operation before the previous read operation completed.
EIONO$DATA	0055H	The tape drive attempted to read the next record, but it found no data.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.

E$LIMIT	0004H	At least one of the following is true: • Either the calling task's job or the job's default user object is already involved in 255 (decimal) I/O operations. • The calling task's job is not an I/O job.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a file connection.
E$PARAM	8004H	The function code is not a legitimate value.
E$SPACE	0029H	At least one of the following is true: • The call attempted to format a track that is beyond the end of the volume. • When formatting a RAM-disk or a tape, the call attempted to format a track other than track zero.
E$STREAM$SPECIAL	003CH	At least one of the following is true: • The calling task is attempting to satisfy a stream file request, but there is no request queued at the stream file. • The calling task attempted to satisfy a stream file request, but the only queued request is a query. • The calling task is querying a stream file, but the only request queued at the file is another query. The Extended I/O System removes both queries from the queue and returns this exception code.
E$SUPPORT	0023H	The specified connection was created by a task outside of the calling task's job.

The S$TRUNCATE$FILE system call removes information from the end of a named data file. This system call can be used only with named files.

```
CALL RQ$S$TRUNCATE$FILE(connection, except$ptr);
```

Input Parameter

connection	A TOKEN for a connection to the named data file that is to be truncated. The file pointer for this connection tells the Extended I/O System where to truncate the file. The byte indicated by the pointer is the first byte to be dropped from the file.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns a condition code.

Description

This system call applies to named data files only. When called, it truncates a file. "Truncate" means to get rid of the data in the file from the current location of the file pointer to the end of the file.

Unless the file pointer is already where you want it, your task should use the S$SEEK system call to position the pointer before using the S$TRUNCATE$FILE system call.

Truncation will occur immediately, regardless of the status of other connections to the same file. If the pointer is at or beyond the end-of-file, no truncation occurs.

File pointers for other connections to the file are not affected by the truncation operation. Thus, it is possible that file pointers for other connections to the file will be beyond the new end-of-file after the S$TRUNCATE$FILE call. If a task invokes the A$READ or S$READ$MOVE system calls with a file pointer beyond the end-of-file, the Basic I/O System behaves as though the reading operation began at the end-of-file. If a task invokes the A$WRITE or S$WRITE$MOVE system calls with a file pointer beyond the end-of-file, the Basic I/O System attempts to expand the file. If the Basic I/O System does expand your file in this manner, the file contains random information between the old end-of-file and the point in the file where the write begins.

Access Requirements

Three access requirements pertain to this system call. First the connection must be open for writing only or for both reading and writing. If this is not the case, your task can use the S$OPEN system call to open the connection.

Second, the connection must have update access to the file. Recall that the Extended I/O System computes a connection's access when the connection is created.

Third, the connection must have been created by a task within the calling task's job. If this is not the case, use the existing connection as a prefix, and have the calling task invoke the S$ATTACH$FILE system call.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$CONN$NOT$OPEN	0034H	At least one of the following is true: • The connection is open in the wrong mode. It must be open for writing or for both reading and writing. • The connection is not open. • The connection was opened by an A$OPEN rather than an S$OPEN.
E$FACCESS	0026H	The connection does not have update access to the file.
E$EXIST	0006H	The connection parameter is not a token for an existing object.
E$IFDR	002FH	Your task is attempting to truncate a stream or physical file. The S$TRUNCATE$FILE system call can be used only on named files.
EIOHARD	0052H	A hard I/O error occurred. A retry is probably useless.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.

E$LIMIT	0004H	At least one of the following is true: • The calling task's job is not an I/O job. • Either the calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a file connection.
E$SPACE	0029H	The truncation required writing the contents of a buffer to the file, but the volume was full.
E$SUPPORT	0023H	The connection was created by a task outside the calling task's job.

S$UNCATALOG$CONNECTION

The S$UNCATALOG$CONNECTION deletes a logical name from the object directory of a job.

```
CALL RQ$S$UNCATALOG$CONNECTION(job, log$name$ptr, except$ptr);
```

Input Parameters

job	A TOKEN for a job. The Extended I/O System deletes the logical name from this job's object directory. Setting the job parameter to SELECTOR$OF(NIL) specifies the calling task's job.
log$name$ptr	A POINTER to a STRING (of 1 to 12 characters) containing the logical name to uncatalog. The name can be delimited with colons (:). The operating system removes the colons so that a logical name with colons is the same as one without (e.g., :F0: is effectively the same as F0). Colons do not count in the length of the name.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

Your tasks should invoke this system call to delete logical names that were added to the object directory by the S$CATALOG$CONNECTION system call.

Condition Codes

E$OK	0000H	No exceptional conditions.
E$EXIST	0006H	The job parameter is not a token for an existing object.
E$LIMIT	0004H	The calling task's job is not an I/O job.
ELOGNAME$-NEXIST	0045H	The call could not find the logical name in the job's object directory.

ELOGNAME$-SYNTAX	0040H	The syntax of the specified logical name is incorrect because at least one of the following conditions is true: • The STRING pointed to by the log$name$ptr parameter is of zero length or has a length greater than 12 (not including colons (:)). • The logical name contains invalid characters.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
E$TYPE	8002H	The job parameter is a token for an object that is not a job.

The S$WRITE$MOVE system call writes a collection of bytes from a buffer to a file.

```
bytes$written = RQ$S$WRITE$MOVE(connection, buf$ptr, count,
                                except$ptr);
```

Input Parameters

connection	A TOKEN for the connection to the file in which the information is to be written.
buf$ptr	A POINTER to a contiguous collection of bytes that are to be written to the specified file.
count	A WORD containing the number of bytes to be written from the buffer to the file.

Output Parameters

bytes$written	A WORD containing the number of bytes that were actually written to the file. This number will always be equal to or less than the number specified in the count parameter.
except$ptr	A POINTER to a WORD where the Extended I/O System returns a condition code.

Description

This system call causes the Extended I/O System to write the specified number of bytes from the buffer to the file.

Access Control

To write information into a file, the connection parameter must satisfy the following two requirements:

- The connection must have been created by a task within the calling task's job. If this is not the case, the Extended I/O System returns an E$SUPPORT exception code.
- The connection must be open for writing or for both reading and writing.

If the file is a named data file, the access rights associated with the connection must permit the kind of writing being performed. That is, if you are writing over data in the file, the connection must have update access or you will get an exception code; if you are writing data beyond the end-of-file, the connection must have append access or you will receive an exception code.

The connection can have access rights for updating, appending, or both. For information regarding the process of assigning access to a connection, see the descriptions for the S$ATTACH$FILE and S$CREATE$FILE system calls.

Number of Bytes Actually Written

Occasionally, the Extended I/O System writes fewer bytes than requested by the calling task (upon return from the call, bytes$written is less than count). This happens under two circumstances:

- When the Extended I/O System encounters an I/O error. Your task will be informed of this circumstance because the Extended I/O System returns an exception code.
- When the volume to which your task is writing becomes full. The Extended I/O System informs your task of this condition by returning an E$SPACE exception code.

Where the Bytes are Written

The Extended I/O System writes the first byte starting at the byte pointed to by the file pointer. As the Extended I/O System writes the bytes, it also updates the pointer. After the writing operation is complete, the file pointer points to the byte immediately following the last byte written.

Use the S$SEEK system call to position the file pointer if you are performing random-access operations.

If your task is using a connection that has append access, the task can start a writing operation beyond (rather than at) the EOF. The Extended I/O System extends the file and performs the writing operation. If the file is extended, the extended section of the file contains unknown, random information (you can write data into this area later). For example, if the EOF is at location 200 and your task positions the file pointer at 250 and begins writing, locations 200 through 249 contain undetermined information.

Effects of Priority

The priority of the task invoking this system call can greatly affect the performance of the application system. For better performance, the priority of the invoking task should be equal to or lower than (numerically greater than) 130. If the priority of the calling task is greater than 130, the operating system cannot overlap the write operation with computation or with other I/O operations. (To find out how to set priorities for application tasks, refer to the *iRMX® I Nucleus User's Guide* or the *iRMX® II Nucleus User's Guide*.)

Special Considerations for iRMX®-NET

iRMX-NET's remote file driver does not perform fragmentation and reassembly. For optimal performance, reading and writing should begin at offsets that are integral multiples of the remote server's buffer size. The device$granularity parameter returned by the SGETFILE$STATUS system call indicates the buffer size of a remote server.

Condition Codes

E$OK	0000H	No exceptional conditions.
EBADBUFF	8023H	**This condition code is not supported in the iRMX I Operating System.** One of the following is true: • The specified source memory buffer is not writeable. • The specified source memory buffer crosses segment boundaries.
E$CONN$NOT$OPEN	0034H	At least one of the following is true: • The connection is not open for writing. • The connection is not open. • The connection was opened with A$OPEN rather than with S$OPEN.
E$EXIST	0006H	The connection parameter is not a token for an existing object.
E$FACCESS	0026H	The call tried to write beyond the end-of-file, but the connection specified does not have append access to the file.
E$FLUSHING	002CH	The specified device is being detached.
E$FRAGMENTATION	0030H	The file is too fragmented to be extended.

EIOHARD	0052H	A hard I/O error occurred. Another retry is probably useless.
EIOMODE	0056H	One of the following is true: • A tape drive attempted to perform a read operation before the previous write operation completed. • A tape drive attempted to perform a write operation before the previous read operation completed.
EIOOPRINT	0053H	The device was off-line. Operator intervention is required.
EIOSOFT	0051H	A soft I/O error occurred. The I/O System tried to perform the operation a number of times and failed (the number of retries is a configuration parameter). Another retry might still be successful.
EIOUNCLASS	0050H	An unknown type of I/O error occurred.
EIOWRPROT	0054H	The volume is write-protected.
E$LIMIT	0004H	At least one of the following is true: • The calling task's job is not an I/O job. • The calling task's job, or the job's default user object, is already involved in 255 (decimal) I/O operations.
E$MEM	0002H	The memory available to the calling task's job is not sufficient to complete the call.
ENOTCONFIGURED	0008H	This system call is not part of the present configuration.
ENOTCONNECTION	8042H	The connection parameter is a token for an object that is not a file connection.
E$PARAM	8004H	The calling task is attempting to write beyond the end of a physical file.
E$SPACE	0029H	The volume is full.
E$SUPPORT	0023H	The connection parameter refers to a connection that was created by a task outside of the calling task's job.

The VERIFY$USER system call validates a user's name and password.

```
CALL RQ$VERIFY$USER(user$t, name$ptr, password$ptr, except$ptr);
```

Input Parameters

user$t	A TOKEN for the user object to be verified.
name$ptr	A POINTER to a STRING containing the user name. This name would typically be entered from the console during dynamic logon. Only the first eight characters are used; any additional characters are ignored.
password$ptr	A POINTER to a STRING containing the unencrypted user password. This password would typically be entered from the console at the same time as the name$ptr parameter. Only the first eight characters are used.

Output Parameter

except$ptr	A POINTER to a WORD where the Extended I/O System returns the condition code.

Description

The VERIFY$USER system call validates a non-resident user's name and password. Validation means determining if the name and password supplied as parameters identify a predefined user of an iRMX system. This system call searches the file :CONFIG:UDF (User Definition File) for a matching user name and password. (See the *Operator's Guide To The iRMX® Human Interface* for information on the :CONFIG:UDF file.) The name must have the exact same form as it appears in the UDF for a match to occur. The password parameter is encrypted and then compared to the encrypted version in the UDF. The ID defined in the UDF is also compared with the ID contained in the user object.

If a matching name, password, and ID are found, the user object is modified to indicate the user has been verified. If iRMX-NET is configured into your system and the VERIFY$USER call succeeds, then you also gain access to remote files. (See the *iRMX® Networking Software User's Guide* for more information on iRMX-NET.)

If the name is not found or if the password, once encrypted, does not match the encrypted password associated with the name in :CONFIG:UDF, or if the IDs are not the same, an error is returned and the user object is not modified.

The Human Interface can use the VERIFY$USER system call to check a dynamic logon process.

NOTE

The remote file driver will reject all user tokens created by the CREATE$USER system call unless the VERIFY$USER system call is used to verify the user tokens created.

Condition Codes

E$OK	0000H	No exceptional conditions.
EBADCALL	8005H	A task wrote over the interface library or over the EIOS job.
E$CONTEXT	0005H	The user TOKEN has already been verified.
E$DEVFD	0022H	The device cannot be used with the file driver as specified in the preceding logical attach operation.
E$DEVICE$-DETACHING	0039H	An I/O operation could not be performed on the device because it was being detached.
E$EXIST	0006H	The user TOKEN parameter is not valid.
E$FACCESS	0026H	The user does not have the proper access rights for the requested operation.
E$FLUSHING	002CH	The device is being detached.
E$FNEXIST	0021H	One of the following is true: • The file or a file in its path does not exist. • The specified physical device was not found.
E$FTYPE	0027H	A path component is not a directory file.
E$ILLVOL	002DH	The file driver given in the volume label conflicts with the file driver specified in the preceding logical attach operation.
E$INVALID$FNODE	003DH	The fnode associated with a file is either marked not allocated, or the fnode number is out of range.
EIOHARD	0052H	A hard error occurred; the BIOS cannot retry the request.
EIOMEM	0042H	The BIOS job did not have enough memory to perform the requested function.

EIOOPRINT	0053H	The device is off-line; operator intervention is required.
EIOSOFT	0051H	A soft error occurred and the BIOS has retried the operation and failed; a retry is not possible.
EIOUNCLASS	0050H	An unclassified I/O error occurred.
EIOWR$PROT	0054H	The volume is write protected.
E$LIMIT	0004H	The caller's job is not an I/O job.
ELOGNAME$-NEXIST	0045H	The logical name was not found in the caller's object directory, the global job object directory, or the root job object directory.
ELOGNAME$-SYNTAX	0040H	One of the following was true: • A leading colon in the path name STRING indicated the start of a logical name, but a terminate colon was not found. • The logical name STRING has a length of 0 or more than 12 characters. • The logical name STRING contains invalid characters.
E$MEDIA	0044H	The device associated with the system call is off-line.
E$MEM	0002H	The caller's job does not have enough memory to perform the requested operation.
E$NAME$NEXIST	0049H	The name specified in this call is not defined.
E$NOPREFIX	8022H	The caller's job does not have a default prefix, or it is invalid.
ENOTCONFIGURED	0008H	This call is not part of the present configuration.
ENOTLOG$NAME	8040H	The token referred to by the logical name supplied does not refer to a valid device or file connection.
E$NOUSER	8021H	The caller's job does not have a default user or it is invalid.
E$PARAM	8004H	The name or the password contain invalid characters or the name length is equal to zero.
E$PASSWORD$-MISMATCH	004BH	The password is incorrect.
E$SHARE	0028H	The file cannot be shared using the requested access.

E$TYPE	8002H	The user$t parameter is not a TOKEN for a user object.
EUDFFORMAT	0048H	The UDF is not in the correct format.
EUIDNEXIST	004AH	The user ID present in the user token does not match that specified in the UDF.

INDEX

A

B

C

D

E

F

G

H

I

L

M

S

S (continued)

T

V

W

intel®

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of al Intel product users. This form lets you participate directly in the publication process. Your comment will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of thi publication. If you have any comments on the product that this publication describes, please contac your Intel representative.

1. Please describe any errors you found in this publication (include page number).

2. Does this publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). _____

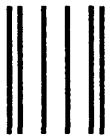
NAME _____ DATE _____
TITLE _____
COMPANY NAME/DEPARTMENT _____
ADDRESS _____ PHONE ( )
CITY _____ STATE _____ ZIP CODE _____
(COUNTRY)

Please check here if you require a written reply. ☐

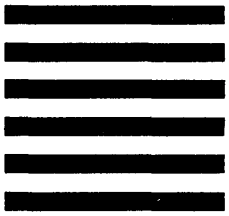
/E'D LIKE YOUR COMMENTS . . .

his document is one of a series describing Intel products. Your comments on the back of this form will elp us produce better manuals. Each reply will be carefully reviewed by the responsible person. All omments and suggestions become the property of Intel Corporation.

you are in the United States, use the preprinted address provided on this form to return your omments. No postage is required. If you are not in the United States, return your comments to the Intel ales office in your country. For your convenience, international sales office addresses are printed on e last page of this document.



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
OMSO Technical Publications, MS: HF3-72
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-9978

INTERNATIONAL SALES OFFICES

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

BELGIUM
Intel Corporation SA
Rue des Cottages 65
B-1180 Brussels

DENMARK
Intel Denmark A/S
Glentevej 61-3rd Floor
dk-2400 Copenhagen

ENGLAND
Intel Corporation (U.K.) LTD.
Piper's Way
Swindon, Wiltshire SN3 1RJ

FINLAND
Intel Finland OY
Ruosilante 2
00390 Helsinki

FRANCE
Intel Paris
1 Rue Edison-BP 303
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL
Intel Semiconductors LTD.
Atidim Industrial Park
Neve Sharet
P.O. Box 43202
Tel-Aviv 61430

ITALY
Intel Corporation S.P.A.
Milandfiori, Palazzo E/4
20090 Assago (Milano)

JAPAN
Intel Japan K.K.
Flower-Hill Shin-machi
1-23-9, Shinmachi
Setagaya-ku, Tokyo 15

NETHERLANDS
Intel Semiconductor (Netherland B.V.)
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam

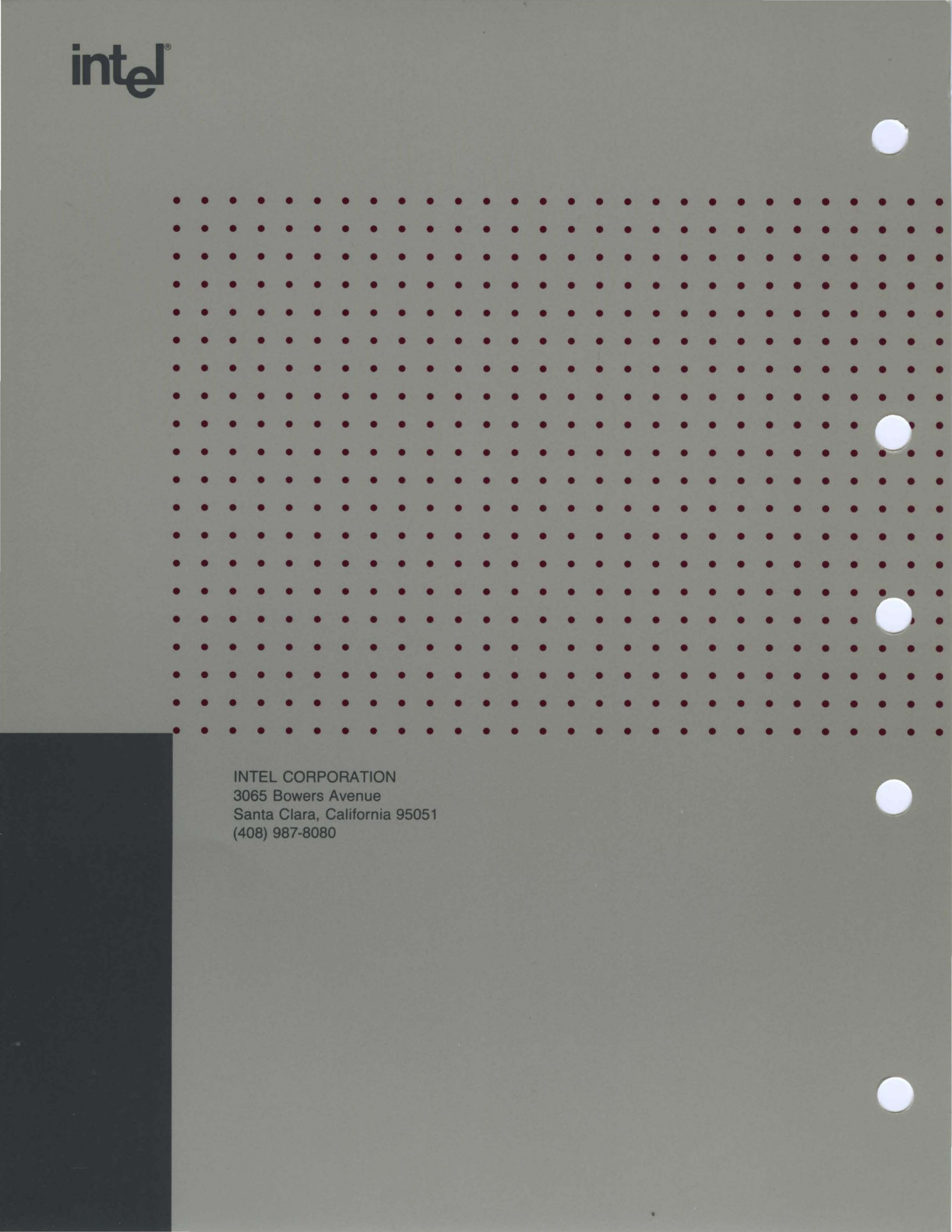
NORWAY
Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013, Skjetten

SPAIN
Intel Iberia
Calle Zurbaran 28-IZQDA
28010 Madrid

SWEDEN
Intel Sweden A.B.
Dalvaegen 24
S-171 36 Solna

SWITZERLAND
Intel Semiconductor A.G.
Talackerstrasse 17
8125 Glattbrugg
CH-8065 Zurich

WEST GERMANY
Intel Semiconductor G.N.B.H.
Seidlestrasse 27
D-8000 Munchen



INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051
(408) 987-8080